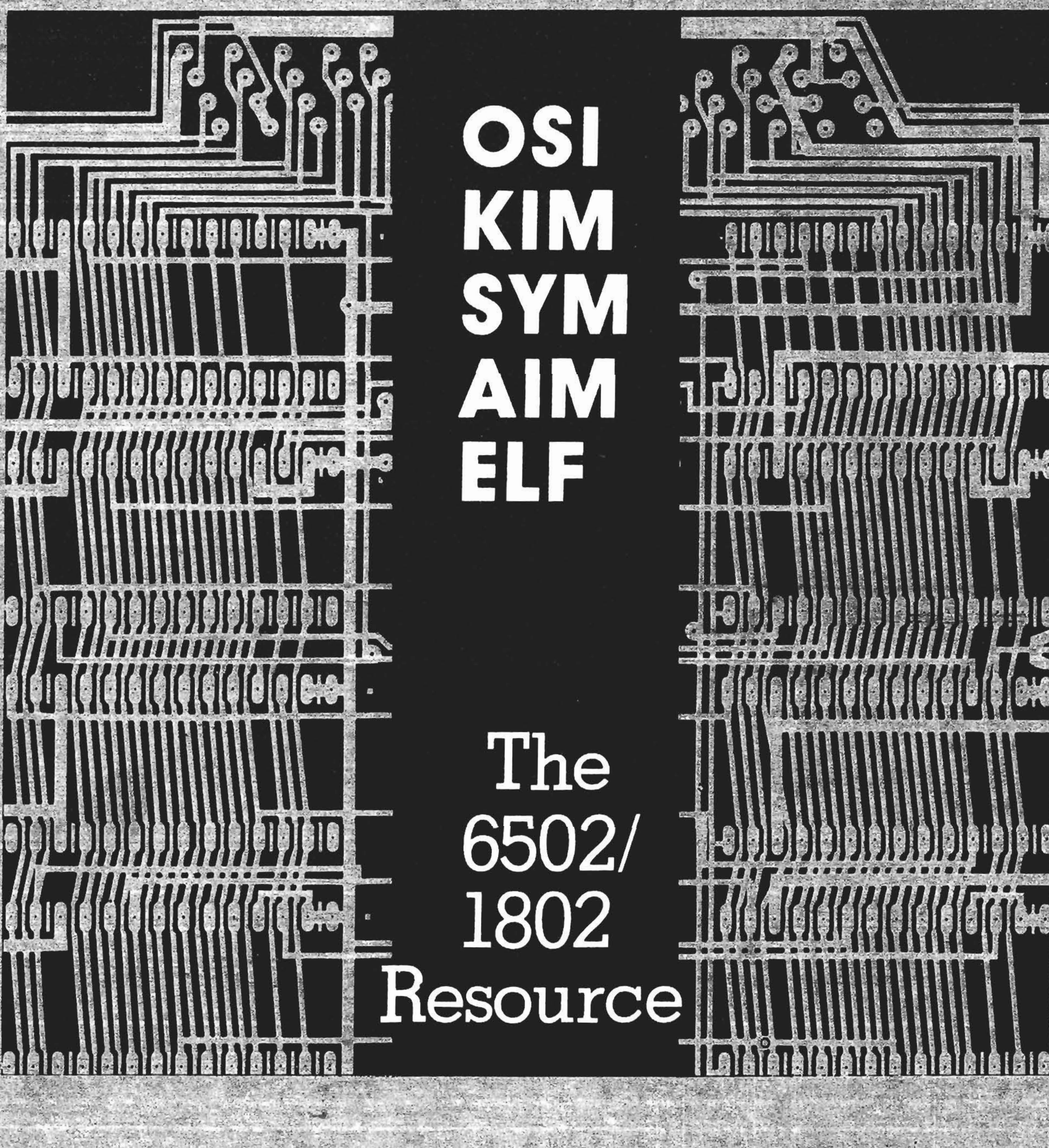
JUNE/JULY, 1980. ISSUE 2. $2.00

compute II.

The Single-Board COMPUTE™



Some A/D And D/A Conversion Techniques

Marvin L. De Jong
Department of Mathematics-Physics
The School of the Ozarks
Pt. Lookout, MO 65726

INTRODUCTION

The purpose of this paper is to describe some A/D conversion circuits and programs that can be used with 6502 based microcomputer systems. A digital-to-analog (D/A) converter is also described. Our motivation for this project was an NSF Short Course on the Science of Sound. The complete project was to be a circuit that would sample some waveform, from an electric guitar for example, and a program that would perform a Fast Fourier Transform (FFT). The Fast Fourier Transform program has not yet been completed, but the necessary A/D circuit and driver programs have been completed and are herein described. A digital-to-analog converter allows the sampled waveform to be displayed on an oscilloscope, producing a much improved storage oscilloscope over our original "storage scope" described in THE BEST OF MICRO, Volume 1, page 30, and Volume 2, page 61. Some results of our experiments are also included.

The analog-to-digital converter is based on the AD570, an 8-bit A/D converter sold by Analog Devices, Route 1 Industrial Park, P.O. Box 280, Norwood, MA 02062. Its nominal conversion time is 25 microseconds, allowing a maximum sampling rate of 40,000 kHz. (The time necessary to read the converter and store the data will reduce this rate.) The AD570 requires *no external components,* and can be operated either in a bipolar or a unipolar mode. We chose it because it is inexpensive, relatively fast, and easy to interface.

The D/A converter we used is a Signetics NE5018. It is also easy to interface because it has input latches. It can be operated with few external components, but it is not an exceptionally fast converter. A better choice would have been the Analog Devices 565, but this would have required an 8-bit latch.

The circuits shown here interface to the expansion connectors on the KIM-1 or the AIM 65. With little modification they could be connected to a SYM-1. The application connector is left free for other devices. In particular, we had hoped to do our mathematics for the FFT program with an AM9511 Arithmetic Processor Unit interfaced to the I/0 ports on the application connector. In any case, Appendix A suggests a circuit for interfacing the converters to a 6522 Versatile Interface Adapter.

Description Of The Circuit

The complete A/D, D/A, and oscilloscope trigger circuitry is shown in Figure 1. This circuit was used to interface the converters to an AIM 65 microcomputer, and all the necessary connections are available at the expansion connector, including the DS9 device select pulse. The same circuit could be used with a SYM-1 if the DS18 device select pulse, available on the SYM-1 expansion connector, were used. In that case the addresses used to activate the various circuits would be $1800 through $1803. In Figure 1 you will notice that addresses $9000 through $9003 produce pulse on the Y_0 through Y_3 outputs on the 74LS138. For example, a STA $9000 instruction produces a negative one microsecond pulse on Y_0. This pulse is applied to the CLEAR input on the 74LS74 flip-flop and it will cause the Q output to go to logic zero. A logic one to logic zero transition on the B/C pin of the analog-to-digital converter (AD570) starts a conversion. Approximately 25 microseconds later the data is ready at the outputs of the AD570. These outputs are connected to an octal, three-state, buffer-driver (81LS97). A LDA $9001 instruction activates the 81LS97 and puts the data on the microcomputer's data bus. The trailing edge of the same device select pulse that enables the 81LS97 clocks the 74LS74 back into its "Q high" logic state. This completes one analog-to-digital conversion.

The analog input voltage is applied to pin 13 of the AD570. The 15 ohm resistor can be omitted if a slight loss of precision is of no concern. With the bipolar offset switch open, as shown in Figure 1, voltages between -5 V and +5 V will be converted to a binary number between $00 and $FF respectively. A binary output of $80 corresponds to pin 13 being at zero volts. If the bipolar output switch is closed, the AD570 will read voltages between 0 V and +10 V. The AD 570 will also work with a negative supply voltage of -12 V rather than the -15 V shown in Figure 1. Although a "data ready" signal is available on the AD570, we chose to use software to wait for the conversion to be completed. One final note on the AD570: the input impedance for the analog input is only about 5 k ohm. Consequently it makes a very poor voltmeter unless a high impedance (a voltage follower circuit, for example) amplifier is placed between the analog input

mode it simply provides a high impedance buffer for the AD570. The AD521 is a differential amplifier with a differential input impedance of about 3 X 10^9 ohms. Pin 3, the - input, need not be grounded but can be connected directly to the input voltage source.

The circuits of Figures 1 - 3 provide a complete A/D and D/A system that can be used for a large variety of applications including voltage measurements, temperature measurements, and the storage scope application described here.

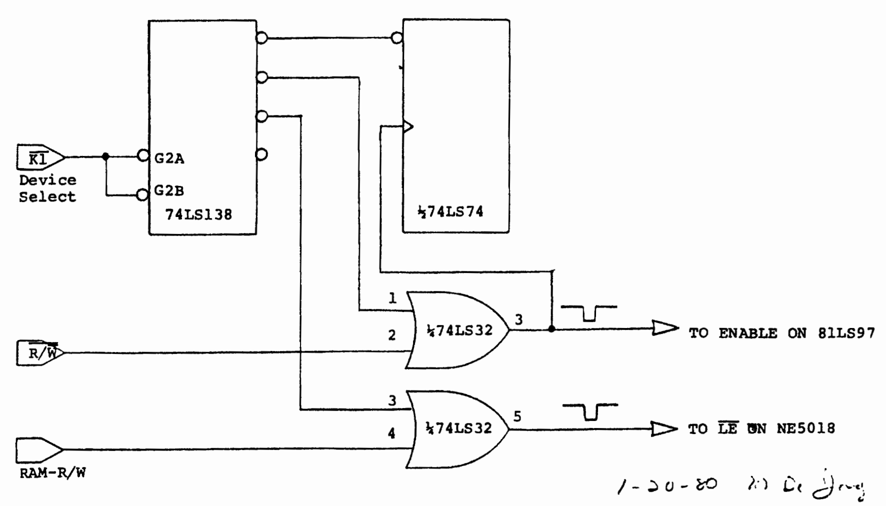
A/D and D/A Converter Software

The program in Example 1 was designed to work with the AIM 65 or any other system that has a 6522 VIA available for timing purposes. The addresses used to start the conversion, read the A/D converter, load the D/A converter, and trigger the oscilloscope are $9000 through $9003, respectively. If a device select other than the DS9 is used to enable the 74LS138 decoder, then these addresses must be changed accordingly. For example, if the DS18 select on the SYM-1 is used, then these addresses become $1800 through $1803, respectively. Since the KIM-1 does not have a 6522, we wrote another program that will work for it, and this program is listed in Example 2.

The program in Example 1 has a maximum sampling rate of one sample every 32 microseconds, or 31,250 Hz. It allows the AD570 exactly 28 microseconds to make a conversion if the T1 timer is loaded with $0000. If you have an AD570 that is slightly faster, try taking out the NOP instruction at $0F3A. If your AD570 is slightly slower, insert some extra NOP instructions after $0F3A. Change the various branch offsets accordingly. You can tell if you are giving the AD570 enough time by examining the data it returns.

The program in Example 1 has the following features. It continuously samples the analog voltage at the input of the A/D

Figure 2. Modifications for the KIM-1.



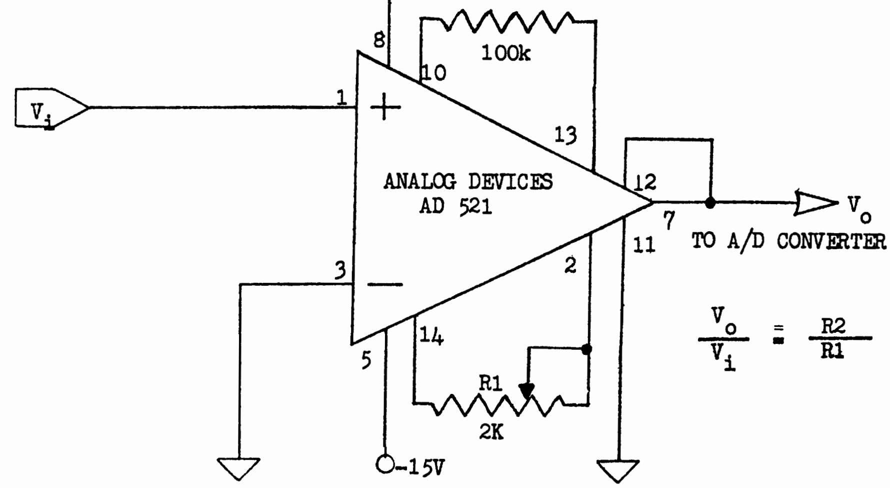
Example 1. A/D and D/A driver program for 6522 based timing.

```
$0F00 8D 00 90  START   STA CVNST   Pulse 74LS74 flip-flop to be in a
                                    known
 0F03 AD 01 90          LDA A/D     condition with Q at logic one.
 0F06 A2 00             LDX $00     Initialize X register to zero
 0F08 A9 40             LDA $40     Initialize ACR of 6522 to put T1 in
 0F0A 8D 0B A0          STA ACR     its free-running mode.
 0F0D A9 00             LDA $00     Clear accumulator.
 0F0F F0 03             BEQ TEST    Branch to start the first conversion.
 0F11 AD 01 90  AGN     LDA A/D     Read A/D converter
 0F14 8D 00 90  TEST    STA CVNST   Start a conversion.
 0F17 8D 02 90          STA D/A     Output A/D to D/A converter.
 0F1A C5 00             CMP TRIG    Compare conversion result with trigger
 0F1C B0 0E             BCS SAMPLE  level. Branch to sample an additional
 0F1E A5 01             LDA TIMLO   255 points if A/D exceeds trigger level.
 0F20 8D 04 A0          STA T1LL    Load 6522 with the number of micro-
 0F23 A5 02             LDA TIMHI   seconds between conversions.
 0F25 8D 05 A0          STA T1LH    Start timer.
 0F28 90 37             BCC AGN     Branch to read A/D.
 0F2A AD 01 90  MORE    LDA A/D     Read A/D
 0F2D 8D 00 90  SAMPLE  STA CVNST   Start sampling waveform.
 0F30 9D 00 02          STA TAB,X   Previous result into table.
 0F33 E8                INX         X keeps track of the number of
                                    conversions.
 0F34 F0 0C             BEQ OUT     When X = 00, 256 conversions are
                                    complete.
 0F36 AD 04 A0          LDA T1CL    Clear T1 interrupt flag.
 0F39 EA                NOP         Fill in the 25 microsecond conversion
 0F3A EA                NOP         time with no operation instructions.
 0F3B 2C 0D A0  LOAF    BIT IFR     Has T1 timed-out?
 0F3E 70 E9             BVS MORE    Yes, get another conversion.
 0F40 50 F9             BVC LOAF    No, wait for timer.
 0F42 8D 03 90  OUT     STA SCPTRG  Trigger scope.
 0F45 EA                NOP         Use an eight microsecond pulse to
 0F46 EA                NOP         trigger scope.
 0F47 8D 03 90          STA SCPTRG
 0F4A BD 00 02  NEXPT   LDA TAB,X   Get some data from the table.
 0F4D 8D 02 90          STA D/A     Output it to the D/A and the scope.
 0F50 E8                INX
 0F51 D0 F7             BNE NEXPT   Branch to get more data.
 0F53 F0 ED             BEQ OUT
$0000 = TRIG; load with desired triggering level but not $00.
$0001 = TIMLO; low-order byte of time interval between samples (microseconds).
$0002 = TIMHI; high-order byte of time interval between samples.
$0200 = TAB: base address of table to store 256 samples.
$9000 = CNVST; a STA CVNST instruction will start an A/D conversion.
$9001 = A/D; the analog-to-digital converter is read at this location.
$9002 = D/A; write to this location to perform a digital-to-analog conversion.
$9003 = SCPTRG; write to this location to trigger the oscilloscope.
```

converter. When the conversion result exceeds a preassigned level stored in TRIG (location $0000), the program proceeds to sample another 255 points on the waveform at a rate determined by the numbers stored in TIMLO (location $0001) and TIMHI (location $0002). The 256 data points are stored in page two of memory. Once the data have been obtained, the program proceeds to read the data out, one point at a time, to the D/A converter for the purpose of displaying the values on an oscilloscope. Each time the 256 points are output to the D/A converter, a trigger pulse is also supplied. Since the conversion time is 32 microseconds with this program, there is no use loading the T1 timer with a number less than 32 unless you wish to sample at the maximum rate. In that case, put $00 in location $0001. In the program in Example 1, T1 is in its free running mode, so its interrupt flag, IFR6, will be set every N + 1 microseconds, where N is the 16-bit number loaded into T1 from locations $0001 and $0002. Be sure to load the locations TRIG, TIMLO, and TIMHI before running the program. The program comments should explain how the program works. The first two instructions may produce a dummy conversion, but their real function is to put the 74LS74 flip-flop in a condition with Q at logic one. The program consists of three main loops. The AGN loop continuously samples the incoming data, and the program branches out of this loop to the MORE loop when the incoming voltage exceeds the trigger level. In the MORE loop another 255 points are produced. When this data has been gathered, the program branches to the OUT loop to output the 256 points to the D/A converter.

The program in Example 2 works in about the same way with the same purpose in mind, but it was used on the KIM-1. The sampling rate with this program is once every 39 microseconds, or 25641 Hz. Its speed could be

Figure 3. Preamplifier circuit for the A/D converter.



Example 2. A/D and D/A driver program for a KIM-1 interface.

0300 8D 00 04	START	STA CVNST	Pulse 74LS74 flip-flop to be in a known condition with Q at logic one.
0303 AD 01 04		LDA A/D	
0306 A2 00		LDX $00	Initialize X register to zero.
0308 A9 00		LDA $00	Initialize accumulator to zero.
030A 8D 00 04	TEST	STA CVNST	Start A/D conversion.
030D 8D 02 04		STA D/A	Previous result into D/A converter.
0310 C5 00		CMP TRIG	Compare conversion result with trigger
0312 B0 16		BCS SAMPLE	level. Branch to sample 256 points if
0314 EA		NOP	voltage exceeds trigger level.
0315 EA		NOP	Delay with no-operation instructions
0316 EA		NOP	until the 25 microsecond conversion
0317 EA		NOP	time is completed.
0318 EA		NOP	
0319 EA		NOP	
031A EA		NOP	
031B AD 01 04		LDA A/D	Read A/D converter.
031E 90 EA		BCC TEST	Branch to start another conversion.
0320 8D 00 04	MORE	STA CVNST	Start an A/D conversion.
0323 9D 00 02		STA TAB,X	Previous result into table.
0326 E8		INX	X keeps track of number of conversions.
0327 F0 13		BEQ OUT	When X = 00, 256 conversions are complete.
0329 A5 01	SAMPLE	LDA TIME	Get time in microseconds from $0001.
032B 8D 04 17		STA TIMER	Store in divide-by-one timer.
032E EA		NOP	Fill in time to make 25 microseconds
032F EA		NOP	before reading A/D converter.
0330 EA		NOP	
0331 EA		NOP	
0332 AD 01 04		LDA A/D	Read converter.
0335 2C 07 17	LOAF	BIT TIMER	Has timer timed out?
0338 30 E6		BMI MORE	Yes, then start another conversion and
033A 10 F9		BPL LOAF	store the result of the last. Otherwise
033C 8D 03 04	OUT	STA SCPTRG	wait. Trigger the oscilloscope.
033F A2 00		LDX $00	
0341 8D 03 04		STA SCPTRG	
0344 BD 00 02	NEXPT	LDA TAB,X	Get some data from the table.
0347 8D 02 04		STA D/A	Output it to the D/A and the oscillo-
034A E8		INX	scope.
034B D0 F7		BNE NEXPT	Branch to get more data.
034D F0 ED		BEQ OUT	Return to output table again.

$0000 = TRIG; load with desired triggering level.
$0001 = TIME: load with time (in microseconds) between samples.
$0400 = CVNST; a STA CVNST instruction will start an A/D conversion.
$0401 = A/D; the analog-to-digital converter is read at this location.
$0402 = D/A; write to this location to perform a digital-to-analog conversion.
$0403 = SCPTRG; write to this location to trigger the oscilloscope.

improved to be about the same as the program in Example 1. In any case, the on-board timers on the KIM-1 were used to produce the necessary timing. Again, the trigger level is stored in $0000, and the time is stored in $0001. The divide-by-one timer at $1704 on the KIM-1 was used, but the other timers may also be used.

The last program listing for the circuit in Figure 1 is a program that can be used to sample a waveform at as many points as your R/W memory will allow. Rather than use just one page of R/W memory for storing the waveform, it will use as many pages as you have available. The maximum sampling rate for this program is one sample every 43 microseconds or 23256 Hz. The program in Example 3 uses several of the same locations as the program in Example 1. The trigger level is stored in TRIG at $0000. The 16-bit number giving the number of microseconds between samples is stored in TIMLO at $0001 and TIMHI at $0002. The low-order byte of the base address of the table to store the conversion data is in location TAB at $0003. Normally this location initialized to $00. The high-order byte of the base address (page number) of the table is stored in TAB + 1 at $0004. For our experiments with the AIM 65 we used pages $02 through $0E. The page number of the last page you wish to fill with data is incremented by one and stored in location END at $0005. Thus if page $0E is the last page to be used to store data, then $0F is stored in END. Load location $0006, STARTP with the same value you put in $0004 if you wish to output all the results to the D/A for display on the oscilloscope.

The program in Example 3 samples an incoming waveform at N*256 points where N is the number of pages used to store the data. It then outputs all of these points to the D/A converter *at the same rate* that it sampled the waveform. If you want to output the results faster, replace the BIT IFR and BVC WAIT instructions at $0f5D with NOPs.

Example 3. N-Page A/D Conversion and Storage Program

```
$0F00 8D 00 90  START    STA CVNST     Pulse 7474 to be in a known condition,
 0F03 AD 01 90           LDA A/D       with Q at logic one.
 0F06 A0 00              LDY $00       Initialize Y to zero for indirect indexed
 0F08 A9 40              LDA $40       addressing that follows.
 0F0A 8D 0B A0           STA ACR       Put 6522 T1 in free-running mode.
 0F0D A9 00              LDA $00       Clear A.
 0F0F 8D 00 90  AGN      STA CVNST     Start a conversion.
 0F12 8D 02 90           STA D/A       Output result to D/A converter
 0F15 C5 00              CMP TRIG      Compare conversion result with trigger
 0F17 B0 21              BCS SAMPLE    level.
 0F19 A5 01              LDA TIMLO     Get low-order byte of time between
                                       conversions.
 0F1B 8D 04 A0           STA T1LL      Result into T1.
 0F1E A5 02              LDA TIMHI     Get high-order byte of time between
 0F20 8D 05 A0           STA T1LH      conversions
 0F23 AD 01 90           LDA A/D       Read A/D converter to get conversion
                                       level.
 0F26 90 E7              BCC AGN       Return to compare with trigger level.
 0F28 8D 00 90  DATA     STA CVNST     Start an A/D conversion.
 0F2B 91 03              STA (TAB),Y   Result of previous conversion into table.
 0F2D C8                 INY
 0F2E D0 0A              BNE EQUAL     Branch around the page number incre-
                                       ment routine.
 0F30 E6 04              INC TAB + 1   Increment page number
 0F32 A5 04              LDA TAB + 1   Now compare it with the ending page
 0F34 C5 05              CMP END       number.
 0F36 90 09              BCC MORE      Fill another page.
 0F38 B0 14              BCS NOMORE    Table is filled, branch to output the table.
 0F3A EA        SAMPLE   NOP           These NOPs equalize the time between
 0F3B EA                 NOP           loading the table when no page boundary
 0F3C EA                 NOP           is crossed and when a page boundary is
 0F3D EA                 NOP           crossed.
 0F3E EA                 NOP
 0F3F A5 05              LDA TAB + 2   This is also a dummy instruction.
 0F41 AD 04 A0  MORE     LDA T1CL      Clear the T1 interrupt flag.
 0F44 AD 01 90           LDA A/D       Read the A/D converter.
 0F47 2C 0D A0  LOAF     BIT IFR       Has the timer timed-out?
 0F4A 70 DB              BVS DATA      Start another conversion.
 0F4C 50 F9              BVC LOAF      Wait for timer.
 0F4E 8D 03 90  NOMORE   STA SCPTRG    Trigger scope.
 0F51 A5 06              LDA STARTP    Reload TAB with starting page number.
 0F53 85 04              STA TAB + 1
 0F55 AD 04 A0  RPT      LDA T1CL      Clear T1 interrupt flag.
 0F58 B1 03              LDA (TAB),Y   Get data from the table.
 0F5A 8D 02 90           STA D/A       Output it to D/A.
 0F5D 2C 0D A0  WAIT     BIT IFR       Test T1 flag.
 0F60 50 FB              BVC WAIT
 0F62 C8                 INY
 0F63 D0 F0              BNE RPT       Get some more data for the D/A converter.
 0F65 E6 04              INC TAB +1
 0F69 C5 05              CMP END
 0F6B 90 EA              BCC RPT       Get more data from a new page.
 0F6D B0 E1              BCS NOMORE    Output the table again.
```

We used this program to see how the waveform from a plucked guitar string varied with time, but we couldn't help connecting a microphone to the AD521 and using the program to output this speech sound to an audio amplifier. The results are quite good, even though we made no attempt to include low-pass filters in either the input or output circuits. The word spoken into the microphone and output to an audio amplifier is intelligible and one can easily identify the person who made the sound. We had enough storage capability on the AIM 65 to store one three-syllable word. If you want a project, you might try improving the circuit and program to do a better job with speech.

Results

In Figure 4 we show a photograph of the results of sampling a 1000 Hz sine wave at a rate of about 25,000 Hz. The photograph shows 256 points on the sine wave. Since we did not have a camera for our oscilloscope, the pictures were taken through a Celestron 5'' telescope, placed about 25 ft. from the oscilloscope. Figure 5 shows the scope trace expanded to show just one

cycle of the same waveform in Figure 4. Figure 6 shows 256 points of a 100 Hz sine wave sampled about once every 40 microseconds, while Figure 7 shows 256 points on a 10 Hz sine wave sampled once every 2000 microseconds. Figure 8 is the waveform of the A string of an electric guitar just after being plucked. The sampling rate in this case was about one sample every 85 microseconds. Finally, in Figure 9 we show a sampled version of a 2500 Hz sine wave. Clearly the system still does a pretty good job of reconstructing a 2500 Hz sine wave, but the information in frequencies much above 5000 Hz will be lost. Hopefully these pictures are worth a thousand words.

Figure 4. 256 points on a 1000 HZ Sine wave.

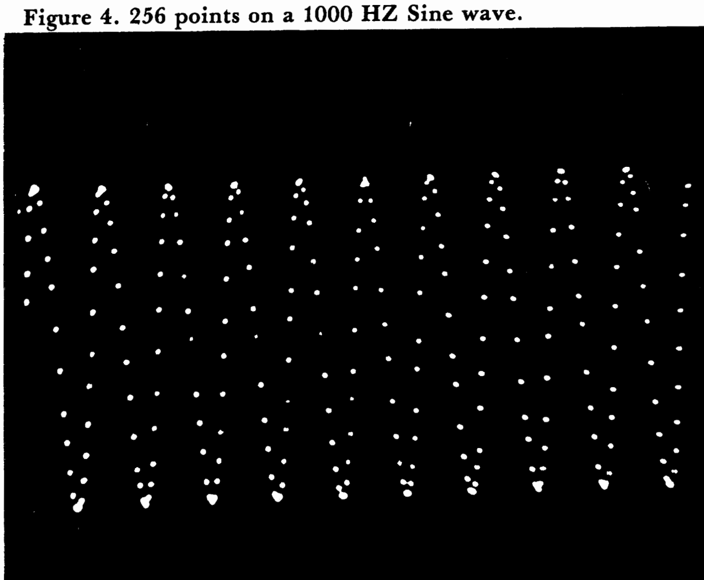


Figure 5. One cycle of a 1000 Hz Sine wave sampled at about 24,000 Hz.

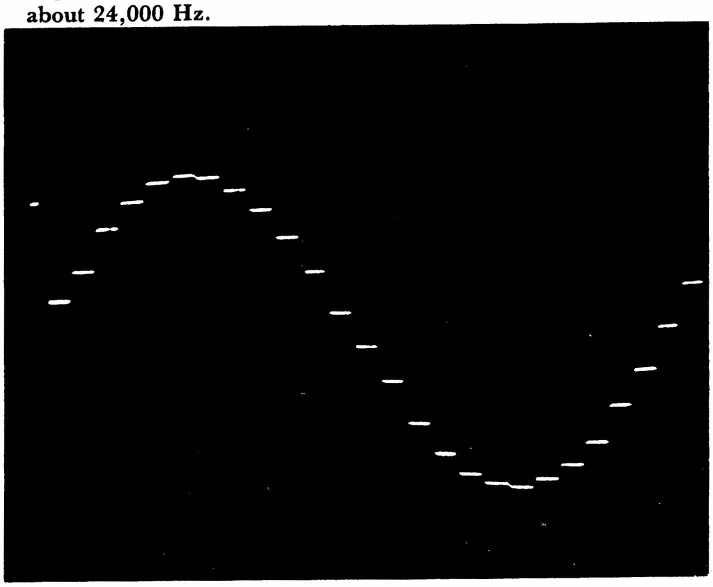


Figure 6. 256 points on a 100 Hz Sine wave.

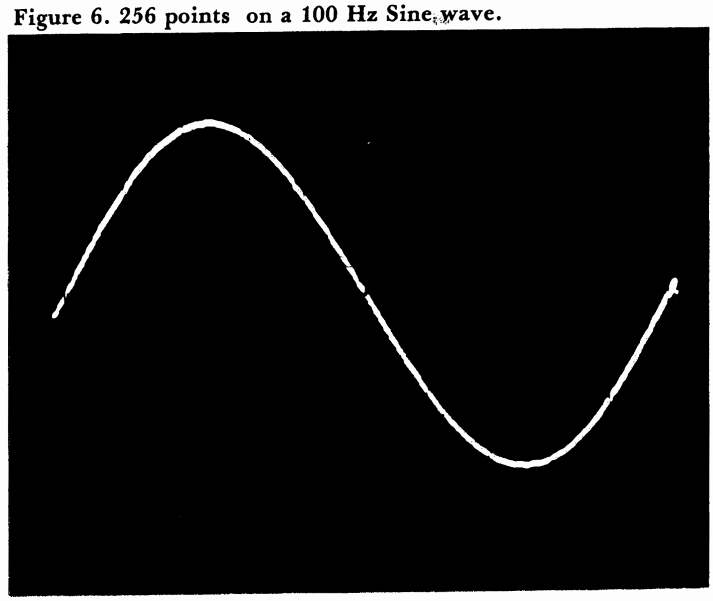


Figure 7. 256 points on a 10Hz Sine wave.

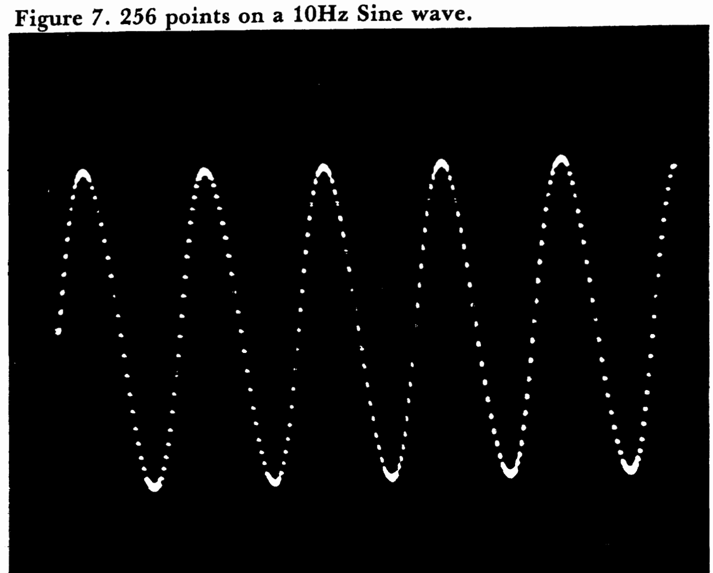


Figure 8. Plucked A string on an electric guitar.

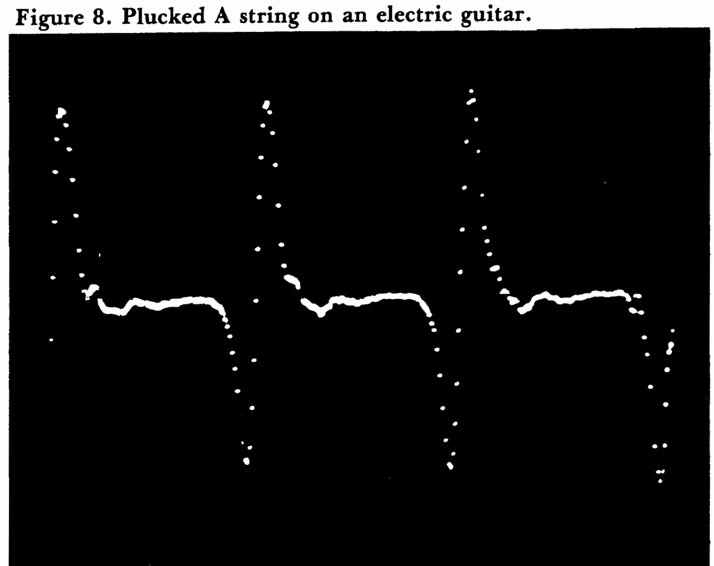


Figure 9. Several cycles of a 2500 Hz Sine wave.

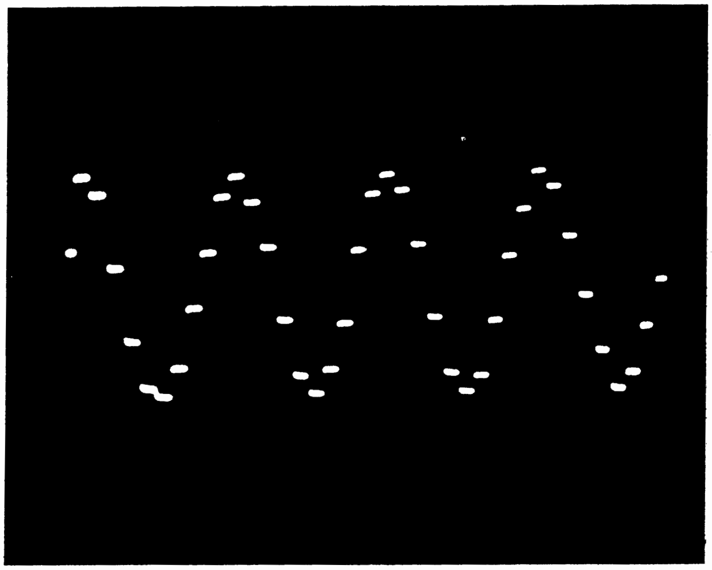
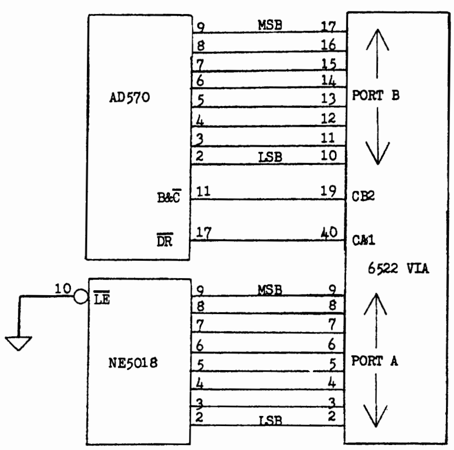


Figure 10. Interfacing the AD570 and the NE5018 to a 6522 Versatile Interface Adapter. Only data and control connections are shown in this figure. Refer to Figure 1 for the other details.



Appendix A. Interfacing The Converters To A 6522 VIA

The AD570 analog-to-digital converter and the NE5018 converter can easily be interfaced to a 6522, eliminating the need for most of the control logic shown in Figure 1. AIM 65 and SYM-1 users may wish to use the 6522 accessed at the application connector and the circuit shown in Figure 10. Note that only the data and control connections are shown in Figure 10. The other circuitry, mainly a few resistors and capacitors, can be found in Figure 1, as are the necessary power connections. This circuit eliminates the 74LS138, the 74LS74, the 81LS97, and the various NAND, NOR, and INVERTER chips. The CA2 pin on the 6522 could be used as an output to trigger the oscilloscope. Below find a short assembly language program that will collect 256 conversions and store them. This program has not been tried.

```
        LDA $FF      Set up Port A as an output port.
        STA DDRA
        LDA $18      Set up the ACR so the shift register shifts out (on CB2)
        ORA ACR      at the clock rate.
        STA ACR
        LDA $FE      Set up the PCR so a negative transition on CA1 sets
        AND PCR      its interrupt flag.
        STA PCR
HERE    LDA $03      The shift register is used to supply a 4 microsecond
        STA SR       pulse to the A/D converter to start a conversion.
        LDA $02      Test to see if conversion is complete by reading IFR1.
TEST    AND IFR
        BEQ TEST
        LDA IRB      Read the A/D converter.
        STA TAB,Y    Store the result in a one page table.
        INY
        BNE HERE     When Y = 0, 256 conversions are complete.
                     Otherwise get another conversion.
OUT
```

©

Some Routines From Microsoft Basic

Jim Butterfield, Toronto

Routines were identified by examining specific machines. There may well be other versions of Basic on these machines; the user is urged to exercise caution.

OSI is from a C2-4 machine. KIM is a cassette tape version. SYM and AIM are the ROM versions.

The addresses given identify the start of the area in which the described routine lies. This may not be the proper program entry point or calling address.

KIM	SYM	AIM	OSI	Description
2000	C003	B00A	A000	Action addresses for primary keywords
203A	C03D	B044	A038	Action addresses for functions
2068	C06B	B072	A066	Hierarchy and action addresses for operators
2086	C089	B090	A084	Table of Basic keywords
2169	C16E	B175	A164	Basic messages, mostly error messages
2274	C1AB	B1AC	A1A1	Search the stack for FOR or GOSUB activity
22A2	C1D9	B1DA	A1CF	Open up space in memory
22E5	C21C	B21D	A212	Test: stack too deep?
22F2	C229	B22A	A21F	Check available memory
231F	C256	B257	A24C	Send canned error message, then:
2348	C27E	B27F	A274	Warm start; wait for Basic command
236A	C2A0	B29D	A295	Handle new Basic line input
23F1	C32C	B329	A32E	Rebuild chaining of Basic lines
2420	C359	B356	A34B	Receive line from keyboard
2466	C39F	B3AE	A3A6	Crunch keywords into Basic tokens
24F2	C427	B436	A432	Search Basic for given line number
2521	C456	B465	A461	Perform NEW
253C	C472	B481	A68C	Perform CLEAR
256B	C49F	B4AE	A4A7	Reset Basic execution to start
2579	C4AC	B4BC	A4B5	Perform LIST
2608	C535	B55C	A556	Perform FOR
26AA	C5DA	B601	A5FF	Execute Basic statement
26CB	C60A	B631	A61A	Perform RESTORE
26DA	C619	B640	A62C	Check stop key
26E8	C622	B65C	A638	Perform STOP or END
2711	C64B	B685	A661	Perform CONT
272B	C665		A67B	Perform NULL
273C	C676	B69F	FFF7	Perform SAVE
278C	C6B7		FFF4	Perform LOAD
		B6AB		Special AIM input routines
27CA	C707	B6EC	A691	Perform RUN
27D5	C712	B6F7	A69C	Perform GOSUB
27F2	C72F	B714	A6B9	Perform GOTO
281F	C75C	B741	A6E6	Perform RETURN, then:
2845	C782	B767	A70C	Perform DATA: skip statement
2853	C790	B775	A71A	Scan for next Basic statement
2857	C793	B778	A71D	Scan for next Basic line
2875	C7B2	B797	A73C	Perform IF, and perhaps:
2888	C7C5	B7AA	A74F	Perform REM: skip line
2898	C7D5	B7BA	A75F	Perform ON
28B8	C7F5	B7DA	A77F	Input fixed-point number
28F2	C82F	B814	A7B9	Perform LET
		B89D		Enable printer
297B	C8B8	B8A9	A829	Perform PRINT
2A13	C94F	B94A	A8C3	Print string from memory
2A35	C971	B967	A8E0	Print single format character
2A59	C991	B988	A904	Handle bad input data
2A7E		B9AD		Perform GET
2A8D	C9B0	B9BC	A923	Perform INPUT
2AB0	C9DC	B9E7	A946	Prompt and receive input
2AB9	C9E5	B9F0	A94F	Perform READ
2BA2	CAB4	BADC	AA1C	Canned Input error messages
2BC6	CAD8	BB00	AA40	Perform NEXT
2C34	CB43	BB59	AAAD	Check type mismatch
2C48	CB57	BB7F	AAC1	Evaluate expression
2D82	CC9F	BCB9	ABF5	Evaluate expression within parentheses
2D88	CCA5	BCBF	ABFB	Check parenthesis, comma
2D99	CCB6	BCD0	AC0C	Syntax error exit
2D9E	CCBB	BCD5	AC11	Setup for functions
2DA5	CCC2	BCDC	AC18	Variable name setup
2DC5	CCE6	BD00	AC27	Set up function references
2E04	CD25	BD3F	AC66	Perform OR, AND
2E34	CD55	BD6F	AC96	Perform comparisons
2E9F	CE11	BDDA	AD01	Perform DIM
2EA9	CE5F	BDE4	AD0B	Search for variable
2F3D	CEF3	BE78	AD8B	Create new variable
2FA3	CF57	BEDC	ADE6	Setup array pointer
2FB4	CF68	BEED	ADF7	Evaluate integer expression
2FD4	CF8B	BF10	AE17	Find or make array
3181	D138	C0BD	AFAD	Perform FRE, and:
3195	D14C	C0D1	AFC1	Convert fixed-to-floating
31A2	D159	C0DE	AFCE	Perform POS
31A8	D15F	C0E4	AFD4	Check not Direct
31B2	D16C	C0F1	AFDE	Perform DEF
31E0	D19A	C11F	B00B	Check FNx syntax
31F3	D1AD	C132	B01E	Evaluate FNx
3266	D21E	C1A3	B08C	Perform STR$
3276	D22E	C1B3	B09C	Do string vector
3288	D240	C1C5	B0AE	Scan, set up string
32EF	D2A9	C232	B115	Build descriptor
3321	D2DB	C264	B147	Garbage collection
3434	D3F2	C37B	B24D	Concatenate
3471	D42F	C3B8	B28A	Store string
349A	D458	C3E1	B2B3	Discard unwanted string
34D2	D490	C419	B2EB	Clean descriptor stack
34E3	D4A1	C42A	B2FC	Perform CHR$
34F7	D4B5	C43E	B310	Perform LEFT$
3523	D4E1	C46A	B33C	Perform RIGHT$
352E	D4EC	C475	B347	Perform MID$
3556	D516	C49F	B36F	Pull string data
3573	D531	C4BA	B38C	Perform LEN
3579	D537	C4C0	B392	Switch string to numeric
3582	D540	C4C9	B39B	Perform ASC
3592	D550	C4D9	B3AB	Get byte parameter
35A4	D562	C4EB	B3BD	Perform VAL
35E3	D5A1	C52A	B3FC	Get two parameters for POKE or WAIT
35EF	D5AD	C536	B408	Convert floating-to-fixed
3605	D5C3	C54C	B41E	Perform PEEK
3610	D5DA	C563	B429	Perform POKE
3619	D5E3	C56C	B432	Perform WAIT
3635	D5FF	C588	B44E	Add 0.5
363C	D606	C58F	B455	Perform subtraction
364E	D618	C5A6	B467	Perform addition
3765	D6FD	C686	B537	Complement accum#1
379C	D734	C6BD	B564	Overflow exit
37A1	D739	C6C2	B569	Multiply-a-byte
3802	D772	C6FB	B59C	Constants
3830	D7A0	C729	B5BD	Perform LOG
386E	D7DE	C76A	B5FB	Perform multiplication
3904	D842	C7CB	B64D	Unpack memory into accum#2
392F	D86D	C7F6	B673	Test & adjust accumulators
394C	D88A	C813	B690	Handle overflow and underflow
395A	D898	C821	B69E	Multiply by 10
3971	D8AF	C838	B6B5	10 in floating binary
3976	D8B4	C83D	B6B9	Divide by 10
3987	D8C5	C846	B6CA	Perform divide-by
398C	D8CA	C851	B6CF	Perform divide-into
3A1A	D958	C8E1	B74B	Unpack memory into accum#1
3A3F	D97D	C906	B76B	Pack accum#1 into memory
3A74	D9B2	C93B	B79B	Move accum#2 to #1
3A84	D9C2	C94B	B7AB	Move accum#1 to #2
3A93	D9D1	C95A	B7BA	Round accum#1
3AA3	D9E1	C96A	B7CA	Get accum#1 sign
3AB1	D9EF	C978	B7D8	Perform SGN
3AD0	DA0E	C997	B7F5	Perform ABS
3AD3	DA11	C99A	B7F8	Compare accum#1 to memory
3B13	DA51	C9DA	B831	Floating-to-fixed
3B44	DA82	CA0B	B862	Perform INT
3B6B	DAA9	CA32	B887	Convert string to floating-point
3C0A	DB3B	CABD	B912	Get new ASCII digit
3C3F	DB70	CAF2	B947	Constants
3C4E	DB7F	CB01	B953	Print IN, then:
3C55	DB86	CB0C	B95A	Print Basic line #
3C69	DB9A	CB1C	B96E	Convert floating-point to ASCII
3D99	DCCA	CC4C	BA96	Constants
3DC2	DCF3	CC75	BAAC	Perform SQR
3DCC	DCFD	CC7F	BAB6	Perform power function
3E05	DD36	CCB8	BAEF	Perform negation
3E10	DD41	CCC3	BAFA	Constants
3E3E	DD6F	CCF1	BB1B	Perform EXP
3E91	DDC2	CD44	BB6E	Series evaluation
3EDB	DE0C	CD8E	BBB8	RND constants
3EE3	DE14	CD96	BBC0	Perform RND
3F1F		CDD2	BBFC	Perform COS
3F26		CDD9	BC03	Perform SIN
3F6F		CE22	BC4C	Perform TAN
3F9B		CE86	BC78	Constants
3FD3			BC99	Perform ATN
4003			BCC9	Constants
4041	DE50	CE86	BCEE	CHRGET sub for zero page

Remaining routines are Basic startup.

Nuts and Volts

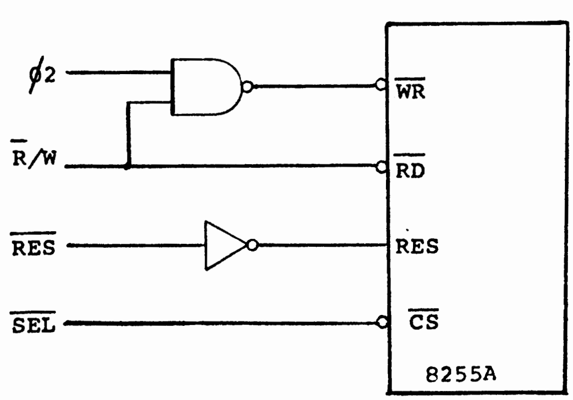
Gene Zumchak
1700 Niagara Street
Buffalo, N.Y. 14207

In the first N & V discussion, I talked about read/write timing in general, and 6502 timing in particular. Fast TTL chips can be used with the 6502, but so can most of the I/O chips of other processor families, provided all the timing requirements are resolved. Even chips with apparently incompatible timing requirements can usually be accomodated by using tricks like latching the write data, or shortening the write strobe, as discussed in the first column. Let's consider what is required to interface a popular port chip of the 8080 family.

The 8255A port chip has 24 I/0 pins, programmable in groups of four or eight bits as inputs and outputs. The ports can be used as simple ports, ports with handshaking (and interrupts) and even as bidirectional buses. The reader might want to dig up a spec sheet to study this versatile chip. The "A" suffix of the part number is important. The original 8255 (without the "A") had long set-up and hold time requirements. The 8255A, like newer Intel family chips, has improved timing specs with a 100 ns. set-up time and 30 ns. hold time, completely 6502 compatible.

The low-true read gate of the 8255A, RD, can be the inverted R/W signal which need not (but can be) gated with Ø2. The low-true write strobe, WR, is met by the normal 6502 write strobe, which we saw earlier is Ø2 NANDed with the inverted R/W line. A high true Reset signal must be provided. Like most peripherals, it has a low-true chip select. Figure 1. shows the connections which satisfy the 8255A's timing requirements.

Figure 1.



Interfacing 8255A Port

If you have an I/0 application requiring more than 16 pins, or you covet some other 8255A feature, there's no electronic reason why you cannot use this chip with your 6502 system. The same can be said of I/0 chips from other families. Clearly, all families are designed to be both voltage level and drive compatible with TTL and hence with each other. As we can see, accommodating the read/write timing need not be difficult.

Using Port Chips

The most commonly used family accessory chips are the I/0 port chips. However, when simple I/0 is required, port chips may not be the best choice. Family chips, including port chips, are not inexpensive. Port chips typically sell in the $8 to $15 range. Since they are MOS devices, their drive capability is usually just one TTL load. They are also vulnerable to static. Since their data bus lines also can only drive a single TTL load, their use is limited to the internal unbuffed data bus around the processor. One could interface them to a buffered bus with bidirectional buffers, but these buffers are expensive and power hungry. MOS port chips, therefore, are most attractive for use in small dedicated controllers, especially where power and parts count are important considerations.

In applications using a buffered bus, where simple I/0 is adequate, and where ruggedness and drive capability are important, TTL I/0 is more attractive, and usually much cheaper.

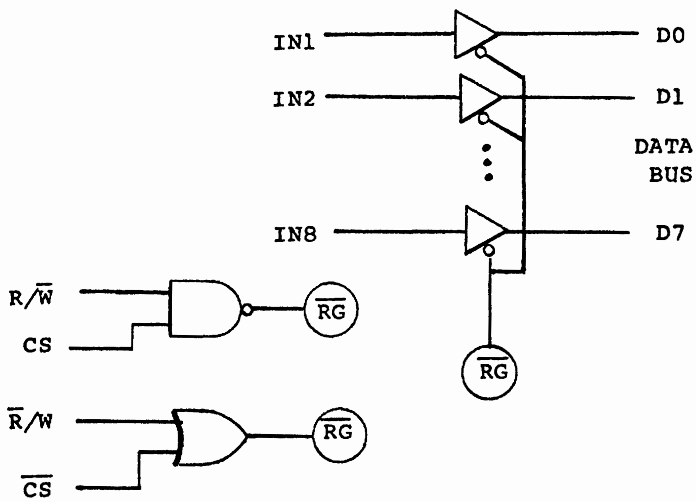
TTL Input

To make an input port, we need a set of tri-state® (*® trademark National Semiconductor)* gates which are gated in unison. A tri-state gate is an electronic switch. When enabled, the output reflects the input (sometimes with inversion). When disabled, the output is high impedance or floating. Thus a large number of tri-state outputs can be bussed together, provided that only one set or device is enabled at one time. RAM chips, ROM chips, and any other devices designed to attach directly to a bidirectional data bus have built in tri-state outputs.

TTL tri-state gates come in quad, hex, and octal configurations. Quad types like the 74LS125 have individual enables for each gate. Hex types like the 8T97, 74LS367, 8097 etc. have four gates with one enable, and two gates with one enable. Although octal gates are the most attractive for eight bit processors, the supply has not kept up with the demand, and hex types are a little easier to come by. Octal types 81LS97 (Nat.) and 74LS244 are not pin compatible.

All that's required to use some tri-state gates as an input port is a low-true read gate. This is obtained by ANDing of the R/W line in the read state, and a chip select decoded from the address lines. Figure 2. shows a couple of possibilities, depending upon the polarity of the chip select.

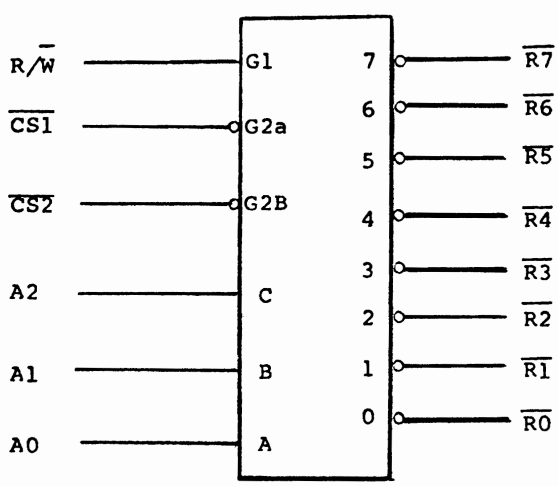
Figure 2.



TTL Input Port With Gating

If read gate signals are required for several ports, a single three to eight decoder chip can be used to get eight read gates from a coarser select. The R/W line is used as an enable and is internally gated with all the outputs, as shown in figure 3.

Figure 3
74LS138



Input Port Read Selects

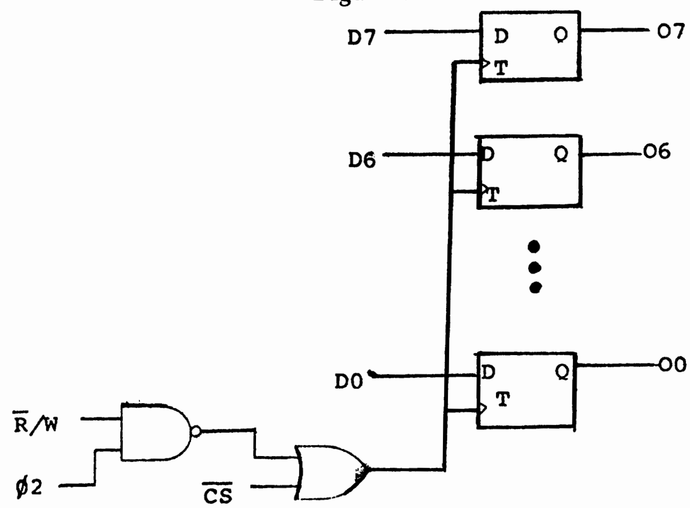
One nice feature of TTL tri-state gates is that they are always buffers and are meant to drive busses. Low power Schottky devices are more desireable and usually adequate for most applications.

TTL Output

An output bit is a flip-flop which can be written to and from the data bus and whose output is connected to the world. Output bits are usually "D" type flip-flops or latches. In TTL there are several configurations, duals, 74LS74, 74LS109; quads, 74LS75, 74LS175; hex, 74LS174; and octal, 74LS273, 74LS373, 74LS374 and others. Again octal types are sometimes a bit hard to come by.

Output ports need a write strobe generated by ANDing the general purpose write strobe with a select decoded from addresses. Figure 4. shows an output port and the necessary write strobe.

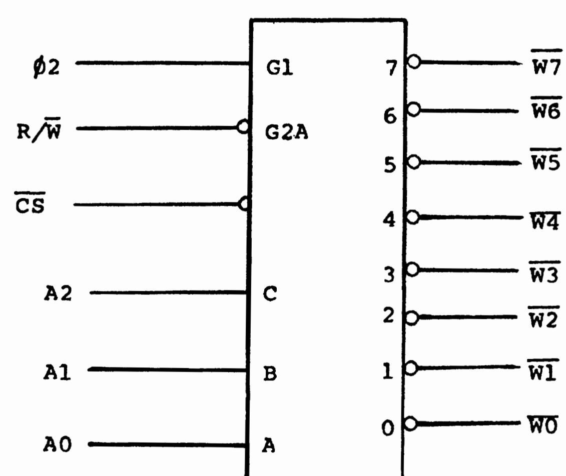
Figure 4.



Output Port With Write Strobe

Since TTL devices are very fast, they have set-up time requirements of only a few nanoseconds. Therefore the locking edge of the write strobe *better* come before the data goes away. That is, the Ø2 closest to the processor must be used, and not any delayed versions. With a little care, we can use a single decoder chip to generate write strobes for several ports as in Figure 5.

Figure 5.
74LS138

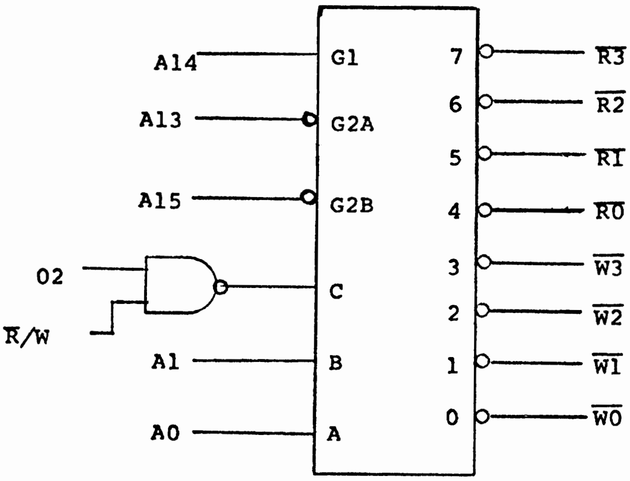


Output Port Write Strobes

From the data sheet for the 74LS138, we see that the delay from the high true enable input (G1) to any output is a maximum of 26 ns. (typically 17 ns.). This is quite acceptible, provided that we are not using a delayed Ø2.

Now if you are building a small dedicated controller, you certainly may not need eight input ports or eight output ports. There's no reason why you cannot use a single 74LS138 to give you gates and strobes for four of each.

Figure 6.
74LS138



Write Strobes and Read Gates

Figure 6 shows a 74LS138 wired to give four read gates and four write strobes. In a dedicated controller, you usually have memory space to burn so that you can afford to waste some. In figure 6. we apply address lines directly to the enables. This puts the ports in an 8K block of memory starting with $4000. The Nand gate generates the general purpose write strobe. It is applied to the "C" input of the decoder. When it is low, a write strobe is generated, when high, a read gate. The maximum delay through the NAND gate is 15 ns, through the decoder, a maximum of 26 ns. Thus Ø2 experiences a worst case delay of 41 ns. to the trailing edge of the write strobe. This would be acceptable even if there was no data bus buffer delay to compensate it.

Summary

Interfacing I/0 to an existing system or a do-it-yourself prototype is not difficult as long as you understand and consider read/write timing. Family chips from any family are useable. Some applications may favor family chips. Others may suggest TTL. The gates and strobes required by TTL I/0 are easy to generate.

In the next column I will talk about address decoding and generating selects. Please feel free to write and suggest hardware topics that you would like me to write about.

©

Programming & Interfacing the 6502, with Experiments,

by Marvin L. Dejong.

Howard W. Sams & Co., Inc.
4300 West 62nd St.
Indianapolis, IN 46268
414 pages, $13.95

Review by Jim Butterfield

This book might have been subtitled, "A hands-on guide to the 6502." That's what it really is: it invites the owner of a KIM, SYM or AIM to learn the 6502 by working through example after example on his machine. Most of us learn by doing, rather than just by reading; and this book contains eighty carefully graded "experiments" that encourage you to get your hands on the machine and prove to yourself that it works the way the book says.

This is good stuff: the text and experiments are carefully graded and go at a gentle pace. You won't get very many advanced programming concepts here: the book covers only the basics. But it does a careful and thorough job. Early concepts are developed with care at a pace the beginner can cope with.

As the title suggests, the book comes in two parts. Part I deals with programming the 6502, Part II with interfacing. Each chapter begins with a statement of objectives, identifying what you may expect to learn there. Each chapter ends with a series of experiments designed to reinforce what you have learned. An experiment often takes the form: "load this program .. now do this .. what do you see? .. can you explain why?". Emphasis is on gaining understanding as to how a simple program operates; the last experiment or two in a chapter often suggest small projects for the reader.

Machine language is developed a few op-codes at a time. Loads, Stores, and Transfers are introduced first, and subsequent chapters progressively bring in more commands. Branches, for example, don't arrive until chapter six - I would rather have seen them a little earlier because I believe loops are so important - and the op codes aren't completely covered until chapter 9 has been completed. Advanced addressing modes, such as indexing and indirect addressing, are held back until chapter 8. It's all carefully graded, and the going is about as easy as it can be for machine language.

The pace changes in Part II, Interfacing the 6502. We're thrown quite abruptly into the hardware field: logic diagrams, truth tables, timing charts and oscilloscope traces start to appear with great rapidity. The author seems to assume that the reader will have some understanding of hardware, which is likely true for a sizable fraction of KIM/SYM/AIM owners. A beginner who isn't sure about the different shapes of AND and NOR logic symbols will have to work hard.

In keeping with the accelerated pace of the material, Part II takes on a number of more ambitious projects, some of which might prove to be of special interest to readers. Music synthesis, an ASCII input port, data logging, a morse keyer, and a lunar occultation program are included; most are adapted from other sources but are accompanied by extra explanations.

The book contains a quite extensive appendix section, with emphasis on hardware. Many of the data sheets are printed in very fine type and may be hard to read. An index is included.

Is it possible to write a book which deals with three different machines--the KIM, SYM, and AIM? The experiments jump around from one machine to the other without always specifying which machine is intended. Even so, most users will be able to sort it out without too much trouble. Two key tables point out vital addresses on the respective machines; readers may find themselves repeatedly scrambling for page 44 or 55 - I wish that these had been printed on fold-out sheets so that they could be visible during the experiments.

The author deals carefully with difficult subjects; he doesn't gloss over the tricky parts but treats them with precision. One thing, however, bothers me: his notation for immediate-mode addressing. If you want to load the A register with the value 12 decimal, any of the following may be used on most assemblers:

```
LDA #12
LDA #$0C
LDA #%00001100
```

.. you may code the number in binary, decimal, hexadecimal or whatever, but you must include that pounds sign (#) to indicate Immediate mode. The author codes LDA $0C; most assemblers would take this to mean, "load the contents of address 12" - not the value 12. Readers will have to re-adapt when they start using an assembler.

The book is a good, gentle introduction to programming the 6502. It's a little harder going for inter-facing, especially for hardware beginners.

The "hands-on" nature of the experiments tend to drive the lessons home. It's a good way to come to grips with your computer.

©

The Single-Board 6502

Eric Rehnke

The 5th West Coast Computer Faire was FANTASTIC!!! Besides having the chance to meet a number of you, I got a real good look at the latest developments in the small computer industry. I am very excited with what's happening.

Everything is becoming increasingly sophisticated. Music, graphics, interface devices, software, applications. . .and on and on.

Graphics seemed to be one dominating theme of the show. Everywhere you looked was evidence on the fact. New and lower cost graphics peripherals were introduced. Two drum plotters for under $700, a graphics input device for $200, sophisticated 3-D software for the Atari machines, graphics animation on the Apple, the list goes on.

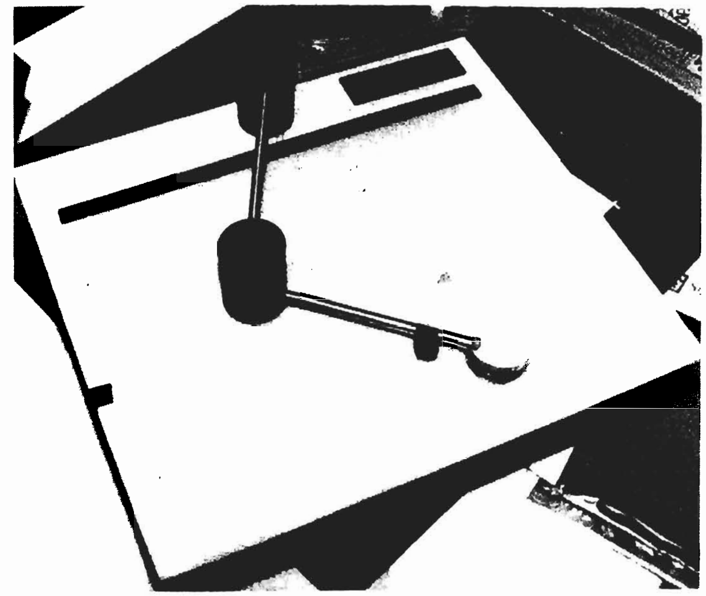
Telecommunications is another area of the industry that is expanding greatly. This is an area which I am particularly interested in because of the fact that as a society, we will be facing an increasing need to replace fossil-fuel burning transportation with energy and time efficient communication. The office of the future will more than likely be in the home for people who can interact with their jobs through a low-cost computer terminal and a modem.

We as computer hobbyists will have much to do with the future of tele-computing. We're the pioneers . . . .

Basically, there are two broad types of information systems accessible today with low-cost equipment. The decentralized type of system includes PCNET (Personal Computer Network), CBBS (Community Bulletin Board Systems) and the like. These systems are fairly casual, since they're more than likely run by hobbyists, have no access charges, and are, at the very least, excellent ways to become familiarized with computer "networking".

The other, more centralized, approach is that taken by The Source and Micronet (to name two). These outfits have large computers with access to very large data bases and many other services available. You can write programs in many of your favorite languages (BASIC, COBOL, FORTRAN, APL, RPG), have access to such things as the UPI General wire service, stock exchange quotes, backgammon, bridge, travel club, a buying service, file generators, editors, Star Trek and Football. On one service you can even download complete programs to your Apple, Pet or TRS-80 (how'd that one get in this column?). Anyhow, all kinds of stuff.

All you need to access this myriad of service is a 300 baud terminal and modem. But, to get the full



For low-cost digital input (about $200), how about this? Your Apple (or whatever) simply reads the position of the two pots which are mounted in the pivot points to compute the position of the arm. Clever, huh???

benefit of all the services, you should also have a microcomputer on your end of the phone line.

Of course, with these large centralized information systems, you have access charges, passwords and the need of a plastic bank credit card to get into the system in the first place. Small price to pay for a little piece of the future, though. Beats the hell outa' the BOOBTUBE!

Getting Hooked Up

Presumably, you already have a computer and a terminal (or a computer with a built-in CRT) and are looking for a modem. The minimum modem necessary will be an originate only, acoustically coupled style capable of handling the BELL 103 standard modem protocol (300 baud). This will permit access to the centralized information system and the hobbyist bulletin board service but will not allow communication with other hobbyists that have orginate-only modems.

You see, for modem systems to communicate with each other, certain conventions must be adhered to. The most important of these states that the system that originates the phone call has to be in the "orginate" mode while the system answering the call should be in the "answer" mode. This originate/answer mode business has to do with the set of frequencies that's used to send the data and need not concern us here except to realize that to be able to receive calls as well as place them, you need both modes (orginate and answer) in your modem system.

Now modems can couple up to the telephone line in two ways: acoustically and directly.

With an acoustically coupled modem, you must usually place the telephone call manually and put the telephone handset into rubber cups on the modem

when the telephone call is connected.

This type of modem is easiest to install, adequate for most applications, and available from several sources in the $150--$200 price range.

If you expect your computer/modem system to be able to automatically answer the phone to carry on a conversation with another system or even be able to automatically place phone calls to other systems without user intervention, you'll want a direct-coupled modem instead of an acoustically-coupled type.

Most direct-coupled modems plug into a modular style phone jack like your extension phone does and allow for full computer control of the line.

Keep in mind that to be completely legal, the modem MUST use a data coupler that has been registered with the FCC guys. Now *that's* important.

Having a fully automatic telephone system hooked to the old computer benefits you in several ways. First, you can take messages from other systems all day long while you're at work or out playing golf (of course, this presumes you have enough friends to make it all worthwhile). And secondly, your computer can place calls to your friendly local (or long distance) data base very late at night to take advantage of low activity and/or cheaper phone rates. You could even download the complete UPI news service to your disk so you can enjoy the up-to-the-minute news with your coffee in the morning. Since the data stream is happening at 300 baud, your computer could sit and scan for key words-picking out only what you're interested in reading about. Quite a bit more efficient than the newspaper. Wouldn't you say?

Anyhow, there are three modem manufacturers which seem interested in supporting the hobby/personal computer market. They are

U. S. Robotics Inc.
1035 W. Lake St.
Chicago, IL 60607
(312) 733-0497

NOVATION Inc.
18664 Oxnard St.
Tarzana, Ca 91356
(213) 996-5060

TNW Corp.
5924 Quiet Slope Dr.
San Diego, Ca 92129
(714) 225-1040

(TNW modem useable only with PET or other IEEE Bus computer)

There are other companies making modems for this market, such as D. C. Hayes but most of these are useable only with certain bus configurations such as Apple or S-100. If you have one of these machines, this part of this column won't prove very useful to you.

I placed a call to U.S. Robotics to get more data on their 300 baud, direct coupled modem and was treated very well. They expressed a willingness to help me with my application and even sent me all their technical literature on the promise that I'd sent them a $5.00 check. No, I didn't tell them that I wrote a column for COMPUTE II. As far as they knew, I was just another hobbyist.

I also had some contact with TNW Corporation. They manufacture stuff for the PET (or other IEEE Bus equipped computers) so their direct-couple modem didn't turn out to be as useful for my particular application. But, if you're looking to turn your PET into an electronic mail system, TNW has the software and hardware to do just that. I believe they are working very closely with the PCNet people so they should have some good software coming out.

As it turns out, the PCNet software protocol is a bit on the complicated side for those of us not well versed in the esoterics of network theory and the like, so having a software package alredy prepared looks mightly appealing.

My personal application for a modem includes use on the PCNet as well as checking into one of the large time sharing systems like the Source or Micro Net (or both). Since I may want to automatically access a data base late at night, the modem/telephone interface needs to be fully automated.

I'll be checking out modems for a while and will report my findings.

Barcodes Come Of Age!!!

Back in 1976 (November to be exact) BYTE magazine introduced an interesting concept regarding program entry from magazine pages (or other printed media).

Using a code very similar to the Universal Product Code, which can be found on just about anything you purchase anymore, programs (and data) can be reproduced on paper in a form that can be fed directly into your computer. This, of course, eliminates, the laborious typing in of magazine software. Just think about the amount of wasted energy when 10,000 computerists across the country have to type in the same program? Now **THAT** amounts to a lot of effort!!! Well, this new scheme could put an end to all that.

I'll bet you're wondering if it's so great, why aren't all the magazines offering software in barcode format. Well, that's a fair question----and the answer is that up until now, bar code reading wands have cost from $300 up.

But that's all changed since Hewlett-Packard introduced the HEDS-3000 bar-code data entry wand for around $100 in single quantities. Now, for a little more than the price of a good audio cassette deck, you can have a truly revolutionary peripheral device for your computer!

Think of all the neat things that can be done with such a device. You computer music users now have the ability to load musical scores directly into your "instrument" (providing of course, music publishing companies print music in some sort of bar code format). Industrial controllers could have the control program or several programs printed right on the face plate for ease of operator input. You could easily input trip data to your car computer or phone numbers to your communication computer. The applications are numerous.

The April 1980 issue of BYTE has an article on the new HP bar code reader and the bibliography of past BYTE articles written on the subject, so I'd suggest you start there if you want more information.

HP can be contacted directly at: 640 Page Mill Rd., Palo Alto, CA 94394 Attention: John Sien.

I'm very tempted to spring for one of these devices but will probably have to put it under the modem on my priority purchase list.

If you'd like to see COMPUTE (or COMPUTE II) publish software in bar code format contact Robert Lock and make yourselves known.

MTU Graphics

I received the Micro Technology Unlimited Visible Memory board a short time ago and have been working on application ideas for this rather unique board.

For those of you not familiar with it: Visible Memory is both an 8K dynamic RAM board with invisible refresh AND a 320x200 bit-mapped video graphics board.

This clever design makes use of the fact that the video circuitry must read the entire 8K block at specified intervals and allows it to serve the double purpose of also refreshing the dynamic RAM. You're wondering why you didn't think of it, right?

"Bit-mapped" means that every bit in the 320x 200 screen matrix is represented by one bit in the screen memory. With this board, one has total control over every pixel. It's very similar to the Apple hi-resolution graphics in that respect, with the exception that the MTU board is slightly denser (320x200 vs. 280x193).

MTU also has some software available for this board that could, assuming you owned an AIM-65, turn your computer into a low cost version of the HP-85. One software package works together with AIM Basic to allow such things as mathematical functions to be graphed out on the display while another software package allows the built-in AIM printer to record whatever pattern is on the screen. How does that sound? That same software also allows text lines up to 80 characters in length to be printed SIDEWAYS on the AIM printer for increased readability.

My appreciation for AIM increased considerably when I saw it performing in this fashion.

Without any further software work, the AIM 65 coupled with some MTU hardware would seem ideally suited for duty in the laboratory, the classroom or most anywhere that a relatively low-cost graphics system can be justified. Assembling such a system turns out to be very easy. It can be performed by someone with moderate electronic skills and with totally "off-the-shelf" components.

But don't let your imagination stop here. Many other things can be done with such a display. How 'bout a 16-channel digital logic analyzer? Very possible with a bit-mapped graphics display.

Want to make your KIM, AIM, or SYM look like a PET? Simple.

PET's screen is organized as 25 lines of 40 characters each. Each of these characters is composed of an 8x8 dot matrix. Multiply 40 characters times 8 bits (character width) and what do you get? Why 320, of course. Then do the same with 25 lines times 8 bits and you get 200.

So, when you break down PET's display to the dot level, the MTU and PET display are precisely the same. It is possible to generate all PET's graphic characters in software or design your own special purpose characters for that matter.

Get the picture?

The Apple and Atari can be simulated in precisely the same fashion. Foreign language fonts are also possible.

Normal X Y plotting subroutines are also in-

cluded in the MTU graphics software.

You can get more information on these and other products from

Micro Technology Unlimited
P.O. Box 4596
Manchester, NH 03108
(603) 627-1464

Sound Chip Update

I finally got hold of some General Instruments Programmable Sound Generator chips (AY3-8910). One of them is residing on a prototype card along with a 6522, which interfaces the sound chip to my computer.

After some initial problems (with me, not the chip) I was able to get the sound generator to start generating some sound. I haven't yet even scratched the surface of what's possible with the PSG-maybe you'll also hook one to your computer and see what sounds you can get out of it.

In my next column, I'll write up the driver software to save you the trouble.

Lately, my mind has also been racing with some of the possibilities for ways to input music into the system as well as output it.

Hope For The OSI Users

There may be hope for you OSI users yet. No, not from OSI but from a company called AARDVARK TECHNICAL SERVICES (1690 Bolton, Walled Lake, MI 48088 tel (313) 624-6316).

They seem to have a really good attitude and sure have lots of low-cost game and utility software for C1 and C2 system users.

Their catalog says it all though with several BASIC program listings (including LIFE), at least 4 pages of useful info on Microsoft BASIC and the OSI system besides the incredibly large catalog of program offerings. Well worth their asking price of $1.

Remember the friend of mine who was working on using his C2-4P as a terminal for his new found love (a KIM-1)? Well, that story had a happy ending when he loaded in the dumb terminal program from AARDVARK and it worked perfectly the first time.

Love those happy endings. ©

Part 2: Implementing the IEEE-488 Bus on a SYM-1

DESIGNING AN IEEE-488 RECEIVER WITH THE SYM

Larry Isaacs, COMPUTE. Staff

This is the second part of an article describing the use of a SYM-1 to interface a PET to a Spinwriter with a serial interface. We will continue to divide the more complex functions into simpler sub-functions when necessary. Once the sub-functions are simple enough, they will be implemented. In the first part, the interface was divided into four sub-functions: INIT, PRINT, CYCLE, and INTERFACE. Implementations for PRINT and CYCLE have already been presented. Briefly, the PRINT routine handles the communication with the Spinwriter. By using the ETX/ACK protocol, the PRINT routine keeps the Spinwriter printing at its maximum speed. The CYCLE routine handles the handshaking necessary to transfer a byte from the IEEE 488 Bus to the SYM. For convenience, these routines are given again in the complete listing of the interface software found at the end of this article. Also, the hardware to connect the Spinwriter to the SYM is shown again in Figure 2.

Before we can begin work on the INTERFACE sub-function, we must first understand how the PET will try to communicate with the SYM using the IEEE 488 Bus. Now we will continue with a description of this communication procedure.

Communicating On The IEEE 488Bus

The next step is to become familiar with how the PET communicates on the IEEE Bus. This discussion will involve two more signal lines. These are the ATN (Attention) line and the EOI (End Or Identify) line.

Each communication on the IEEE 488 Bus can be described as a sequence of three parts. In the first part the PET identifies which device it wishes to communicate with. In the second part it sends or receives the data. And finally in third part, the PET terminates the communication sequence. Each part makes use of the byte transfer cycle described previously to transfer information. However, the information transferred in the first and third parts is differentiated from the second by the state of the ATN line. During the first and third parts the ATN line is low, indicating that the bytes transferred should be treated as commands and not data.

Here is a brief description of what happens during a communication sequence with a device, or devices, which only receives data, such as our printer. I will assume that prior to the beginning of the sequence, all devices on the bus are in the inactive state, i.e. the NRFD line is high.

The sequence begins with the PET setting the ATN line low. This brings all operating devices on the bus to the active state. The PET now executes a byte transfer cycle sending the device address to each device. Only those devices whose device address matches the one sent by the PET will continue with the communication sequence. All other devices will return to the inactive state at the end of this first part. The Commodore printers use device address 24 hex. The lower 5 bits contain the device number, in this case 4. The upper three bits, ''001'', indicate that the device is to receive data. A ''010'' in the upper three bits would indicate the device is to send data. Now the PET may end the first part by setting the ATN line high, or transfer another byte known as the secondary address before setting ATN high. The secondary address is used to address different functions or channels within the selected device.

The second part consists of the required number of byte transfer cycles to transfer the data to the device. In most cases, the PET will signal that the last data byte is being transferred by setting the EOI line low during the last cycle. Because the EOI isn't always sent, it wouldn't be a reliable signal to use for determining the end of this part of the communication sequence.

For the third part, the PET sets the ATN line low again, and executes a byte transfer cycle which sends $3F hex to all active devices. This is the UNLISTEN command, which tells all listening devices to stop receiving data.

One requirement for the interface which may not be obvious is that once the communication sequence has reached the second part, all commands except for the UNLISTEN command should be ignored. It would not be a violation of the IEEE 488 Bus Standard for the PET to activate a device which sends data at the same time as one which receives data, and have them communicate directly with each other.

There is one other IEEE signal line which should be included in the interface. This is the IFC (Interface Clear) line. Whenever this line goes low, the interface should return to the inactive state.

Now we are ready to deal with the hardware requirements for communicating on the IEEE Bus. We will be using 6522 #2 on the SYM for the necessary I/O signals since all of the I/O lines from both ports go to the A-A connector. If necessary, the 6522 supplied as 6522 #3 could be moved to the #2 socket, losing only a few features which aren't needed for

this interface. The main hardware requirement concerns a requirement for the delay between ATN going low to the time when NRFD is set low by a device. The IEEE 488 Standard calls for a maximum of 200 nanoseconds for this delay. Though the PET can't operate this fast, it does operate too fast for the SYM to meet this requirement using just software. The solution to obtain the necessary speed is to selectively send the ATN signal back out the NRFD line. The SYM can then assume control of the NRFD line when it is ready. The only other hardware needed are a couple of open-collector gates for the Wire-or requirements of the NRFD and NDAC lines. The circuitry shown in Figure 1 will meet these requirements.

Interface

The main function of the INTERFACE sub-function is to handle the communication sequence for the IEEE

```
Listing 4

procedure INTERFACE

procedure ATNIRQ begin ... end; {handles the IEEE communication}
procedure IFCIRQ begin ... end; {resets the interface}

begin {INTERFACE procedure}
   repeat
      if INTERRUPT=TRUE then
         begin
            if IRQ=ATN then ATNIRQ;
            if IRQ=IFC then IFCIRQ
         end
   until 2+2=5 {hopefully repeat forever}
end;
```

Bus. The first decision we must make is how the INTERFACE software will know when a communication sequence has begun, or when the IFC line goes low. Since the IFC signal is supposed to reset the device regardless of its current state, this signal should be tied to an interrupt. For greater flexibility we will tie the ATN line to an interrupt as well. This will allow the SYM to do other things when not being used as an interface.

The use of interrupts now provides a basis for dividing the INTERFACE sub-function into smaller parts. Listing 4 shows my division of the INTERFACE sub-function.

At this point we are almost ready to write the assembly language for the remaining parts of the software. However, ATNIRQ needs one more division. This involves addressing the question of how much intelligence to put in the interface. One answer is to program ATNIRQ in a way that leaves the door open for expansion. This can be done easily using the secondary address to call different interface routines. The division for ATNIRQ is shown in Listing 5. The "case" statement in this listing is a multiway subroutine jump. If SECADDRS is 0 when the "case" statement is executed, the SENDASCII procedure will be executed. For other secondary addresses, the DUMPCHRS procedure will be executed.

```
Listing 5

procedure ATNIRQ

procedure ATNINIT; begin ... end; {get ready for communication}
procedure SENDASCII; begin ... end; {input data and print it}
procedure DUMPCHRS; begin ... end; {ignore data}

begin {ATNIRQ statements}
CYCLE; {get device address}
if DATA=MLA then
   begin
      ATNINIT;
      CYCLE; {get next byte, possibly a secondary address}
      if ATN=LOW then
          begin
            SECADDRS := DATA;
            CYCLE
          end;
      case SECADDRS of
            0 : SENDASCII;
         1..15 : DUMPCHRS;
      end {case statement}
   end {if statement}
  end; {ATNIRQ}
```

Now we can write the assembly language for INIT, then IFCIRQ, and finally ATNIRQ. Not clearly shown by the preceeding PASCAL programs is how the machine language should actually handle the interrupts. After an interrupt occurs, the first thing the machine language must do is save the register contents. Then it must test to see what interrupt occured. If it was an ATN interrupt, then the current stack pointer must be saved and ATN interrupts disabled before continuing with the rest of the ATNIRQ routine. If the interrupt was an IFC interrupt, the IFCIRQ routine should test to see if the ATNIRQ routine was executing. If it was, the IFCIRQ routine must restore the stack pointer to the value saved by ATNIRQ and reenable the ATN interrupt before restoring the registers and returning to the interrupted program.

The full listing of the assembly language for the interface is given in Listing 6. I've tried to write the assembly language so it can be easily expanded. Just remember that when you put a different routine in SCTABLE, the first data byte will have already been fetched by CYCLE when your routine is entered.

Summary

I've tried to make this article as much an example of interface design as one describing an actual interface. Most of the material presented dealt with needed facts or the steps involved in reaching a solution. I do not wish to imply that designing an interface should proceed from start to finish as easily as this article makes it seem. It is very likely that during your design, you will come upon a piece of new information or see a different approach which would have been highly useful at some previous step. This occured a few times during this design. Sometimes it is necessary or perhaps desireable to return to that previous step and take a different path. However, if you do enough preparation and planning before you begin the design process, you shouldn't have to backup too many times.

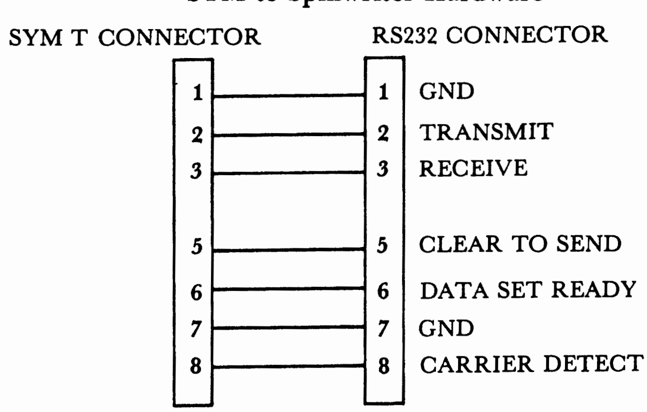
```
                      0010 ; IEEE INTERFACE
                      0020 ; WITH HARDWARE
                      0030 ; VERSION 2.5
                      0040 ;
                      0050 ; CONSTANTS
                      0060 UNLISTEN   .DE $3F
                      0070 BS         .DE $08
                      0080 UNDLN      .DE $5F
                      0090 LF         .DE $0A
                      0100 COLON      .DE $3A
                      0110 SPACE      .DE $20
                      0120 COMMA      .DE $2C
                      0130 CR         .DE $0D
                      0140 ;
                      0150 ; VARIABLES
                      0160 COUNT      .DE $E0
                      0170 SIGNALS    .DE $E1
                      0180 DATA       .DE $E2
                      0190 MLA1       .DE $E3
                      0200 SEC.ADDRS  .DE $E4
                      0210 TEMP       .DE $E5
                      0220 LENGTH     .DE $E6
                      0230 NL.FLAG    .DE $E7
                      0240 SCAN.CNT   .DE $E8
                      0250 F.LEN      .DE $E9
                      0260 SP.IEEE    .DE $EA
                      0270 ;
                      0280 ; ADDRESSES
                      0290 ACCESS     .DE $8B86
                      0300 TOUFL      .DE $A654
                      0310 SDBYT      .DE $A651
                      0320 TECHO      .DE $A653
                      0330 OUTCHR     .DE $8A47
                      0340 INCHR      .DE $8A58
                      0350 CRLF       .DE $834D
                      0360 TOUT       .DE $8AA0
                      0370 @2ACR      .DE $A80B
                      0380 @2DDRA     .DE $A803
                      0390 @2DDRB     .DE $A802
                      0400 @2PCR      .DE $A80C
                      0410 @2IER      .DE $A80E
                      0420 @2IORB     .DE $A800
                      0430 @2IORA     .DE $A801
                      0440 @2IFR      .DE $A80D
                      0450 OUTVEC     .DE $A663
                      0460 UIRQVC     .DE $A678
                      0470 IND.JMP    .DE $EE
                      0480 ;
                      0490            .BA $200
0200- 20 86 8B        0500 INIT       JSR ACCESS    ;INITIALIZATION
0203- A9 24           0510            LDA #$24
0205- 85 E3           0520            STA *MLA1     ;MY LISTEN ADDRESS
0207- A9 90           0530 INIT.SYM   LDA #$90
0209- 8D 54 A6        0540            STA TOUFL     ;ENABLE CRT
020C- A9 10           0550            LDA #$10
020E- 8D 51 A6        0560            STA SDBYT     ;SET FOR 1200 BAUD
0211- A9 00           0570            LDA #$00
0213- 8D 53 A6        0580            STA TECHO     ;OUTPUT & NO ECHO
0216- A9 A0           0590            LDA #L,TOUT   ;SET OUTPUT VECTOR
0218- 8D 64 A6        0600            STA OUTVEC+$1
```

```
0021B- A9 8A      0610              LDA #H,TOUT
021D- 8D 65 A6    0620              STA OUTVEC+$2
0220- A9 53       0630              LDA #L,INTERFACE
0222- 8D 78 A6    0640              STA UIRQVC   ;SET USER IRQ VECTOR
0225- A9 02       0650              LDA #H,INTERFACE
0227- 8D 79 A6    0660              STA UIRQVC+$1
022A- A9 02       0670              LDA #$02
022C- 85 E0       0680              STA *COUNT
022E- A9 00       0690 INITPORTS    LDA #$00
0230- 8D 0B A8    0700              STA @2ACR    ; NO LATCHING
0233- 8D 03 A8    0710              STA @2DDRA   ;2PA7-2PA0 ARE INPUTS
0236- A9 07       0720              LDA #$07
0238- 8D 02 A8    0730              STA @2DDRB   ;3PB2-3PB0 ARE OUTPUTS
023B- A9 04       0740              LDA #$04
023D- 8D 0C A8    0750              STA @2PCR    ;INTERRUPTS
0240- 20 47 02    0760              JSR EN.IEEE  ;ENABLE IRQS
0243- 58          0770              CLI
0244- 4C 44 02    0780 IDLE         JMP IDLE     ;WAIT REAL FAST
                  0790 ;
                  0800 ;
0247- 78          0810 EN.IEEE      SEI
0248- A9 83       0820              LDA #$83     ;ENABLE ATN AND IFC
024A- 8D 0E A8    0830              STA @2IER    ;   INTERRUPTS
024D- A9 06       0840              LDA #$06
024F- 8D 00 A8    0850              STA @2IORB   ;NDAC=1,NRFD=ATN
0252- 60          0860              RTS
                  0870 ;
                  0880 ;
0253- 48          0890 INTERFACE    PHA  ;SAVE REGISTERS
0254- 98          0900              TYA
0255- 48          0910              PHA
0256- 8A          0920              TXA
0257- 48          0930              PHA
0258- AD 0D A8    0940              LDA @2IFR
025B- 10 1D       0950              BPL EXIT.INTF
025D- 29 03       0960 IEEE.IRQ     AND #$03     ;WHICH INTERRUPT?
025F- C9 01       0970              CMP #$01
0261- F0 1D       0980              BEQ ATN.IRQ
0263- C9 02       0990              CMP #$02
0265- F0 03       1000              BEQ IFC.IRQ
0267- 4C 7A 02    1010              JMP EXIT.INTF
026A- AD 01 A8    1020 IFC.IRQ      LDA @2IORA   ;CLEAR INTERRUPT
026D- A9 01       1030              LDA #$01
026F- 2C 0E A8    1040              BIT @2IER    ;IEEE ACTIVE?
0272- D0 06       1050              BNE EXIT.INTF        ;EXIT INTERFACE
0274- A6 EA       1060 IEEE.OFF     LDX *SP.IEEE
0276- 9A          1070              TXS  ;RESTORE STACK POINTER
0277- 20 47 02    1080              JSR EN.IEEE
027A- 68          1090 EXIT.INTF    PLA
027B- AA          1100              TAX
027C- 68          1110              PLA
027D- A8          1120              TAY
027E- 68          1130              PLA
027F- 40          1140              RTI
                  1150 ;
                  1160 ;
0280- BA          1170 ATN.IRQ      TSX
0281- 8E EA 00    1180              STX SP.IEEE  ;SAVE STACK POINTER
0284- AD 01 A8    1190 ATNINIT      LDA @2IORA   ;CLEAR INTERRUPT
0287- A9 05       1200              LDA #$05
```

```
0289- 8D 00 A8                STA @2IORB     ;SET NDAC=0 NRFD=0
028C- A9 01                   LDA #$01
028E- 8D 00 A8                STA @2IORB     ;TURN OFF ATN=NRFD
0291- 8D 0E A8                STA @2IER      ;TURN OFF ATN IRQS
0294- 58                      CLI
0295- A9 00                   LDA #$00
0297- 85 E4                   STA *SEC.ADDRS         ;INIT SEC. ADDRS
0299- 20 EF 02                JSR CYCLE
029C- A5 E2                   LDA *DATA
029E- C5 E3                   CMP *MLA1
02A0- F0 0C                   BEQ DEVICE1    ;BRANCH IF MY ADDRESS
02A2- A9 02       EXIT.IEEE   LDA #$02
02A4- 8D 00 A8                STA @2IORB     ;RELEASE ATN=NRFD
02A7- 2C 00 A8    @15         BIT @2IORB
02AA- 30 FB                   BMI @15        ;WAIT FOR ATN=1
02AC- 10 BC                   BPL IFC.IRQ    ;BR ALWAYS
                  ;
02AE- 20 EF 02    DEVICE1     JSR CYCLE
02B1- 24 E1                   BIT *SIGNALS           ;SECONDARY ADDRESS?
02B3- 10 09                   BPL @3         ;BRANCH IF ATN IS OFF
02B5- A5 E2                   LDA *DATA      ;GET SECONDARY ADDRESS
02B7- 29 0F                   AND #$0F       ;ALLOW 16 SEC.ADDRS'S
02B9- 85 E4                   STA *SEC.ADDRS
02BB- 20 EF 02                JSR CYCLE      ;GET FIRST CHAR.
02BE- A5 E4       @3          LDA *SEC.ADDRS
02C0- 0A                      ASL A
02C1- AA                      TAX
02C2- BD CF 02                LDA SCTABLE,X          ;FIX POINTER TO
02C5- 85 EE                   STA *IND.JMP           ;   SELECTED ROUTINE
02C7- BD D0 02                LDA SCTABLE+$1,X
02CA- 85 EF                   STA *IND.JMP+$1
02CC- 6C EE 00                JMP (IND.JMP)
02CF- 37 03       SCTABLE     .SI SENDASCII          ;NORMAL PRINTING
02D1- 47 03                   .SI DUMPCHRS
02D3- 47 03                   .SI DUMPCHRS
02D5- 47 03                   .SI DUMPCHRS
02D7- 47 03                   .SI DUMPCHRS
02D9- 47 03                   .SI DUMPCHRS
02DB- 47 03                   .SI DUMPCHRS
02DD- 47 03                   .SI DUMPCHRS
02DF- 47 03                   .SI DUMPCHRS
02E1- 47 03                   .SI DUMPCHRS
02E3- 47 03                   .SI DUMPCHRS
02E5- 47 03                   .SI DUMPCHRS
02E7- 47 03                   .SI DUMPCHRS
02E9- 47 03                   .SI DUMPCHRS
02EB- 47 03                   .SI DUMPCHRS
02ED- 47 03                   .SI DUMPCHRS
                  ;
                  ;
02EF- A9 03       CYCLE       LDA #$03
02F1- 8D 00 A8                STA @2IORB     ;NRFD=1 NDAC=0
02F4- 2C 00 A8    @1          BIT @2IORB     ;TEST DAV
02F7- 70 FB                   BVS @1         ;BRANCH IF DAV=1
02F9- 6A                      ROR A
02FA- 8D 00 A8                STA @2IORB     ;NRFD=0 NDAC=0
02FD- AD 01 A8                LDA @2IORA
0300- 49 FF                   EOR #$FF
0302- 85 E2                   STA *DATA
0304- AD 00 A8                LDA @2IORB
```

```
0307- 85 E1     1810           STA *SIGNALS
0309- A9 00     1820           LDA #$00
030B- 8D 00 A8  1830           STA @2IORB    ;NRFD=0 NDAC=1
030E- 2C 00 A8  1840 @2        BIT @2IORB
0311- 50 FB     1850           BVC @2        ;BRANCH IF DAV=0
0313- A9 01     1860           LDA #$01
0315- 8D 00 A8  1870           STA @2IORB    ;NRFD=0 NDAC=0
0318- 60        1880           RTS
0319- 20 47 8A  1890 PRINT     JSR OUTCHR    ;PRINT AND INC. COUNT
031C- E6 E0     1900           INC *COUNT
031E- D0 0C     1910           BNE RETURN
0320- A9 03     1920 ACK       LDA #$03      ;ASCII ETX
0322- 20 47 8A  1930           JSR OUTCHR
0325- 20 58 8A  1940           JSR INCHR     ;WAIT FOR ACK
0328- A9 02     1950           LDA #$02
032A- 85 E0     1960           STA *COUNT
032C- 60        1970 RETURN    RTS
                1980 ;
                1990 ;
032D- A5 E2     2000 @18       LDA *DATA
032F- 29 7F     2010           AND #$7F
0331- 20 19 03  2020           JSR PRINT
0334- 20 EF 02  2030 NEXT      JSR CYCLE
0337- 24 E1     2040 SENDASCII BIT *SIGNALS
0339- 10 F2     2050           BPL @18       ;BR IF ATN=1
033B- A5 E2     2060           LDA *DATA
033D- C9 3F     2070           CMP #UNLISTEN
033F- D0 F3     2080           BNE NEXT
0341- 4C A2 02  2090           JMP EXIT.IEEE
                2100 ;
                2110 ;
0344- 20 EF 02  2120 NEXT2     JSR CYCLE
0347- 24 E1     2130 DUMPCHRS  BIT *SIGNALS
0349- 10 F9     2140           BPL NEXT2
034B- A5 E2     2150           LDA *DATA
034D- C9 3F     2160           CMP #UNLISTEN
034F- D0 F3     2170           BNE NEXT2
0351- 4C A2 02  2180           JMP EXIT.IEEE
0354- 00        2190           .BY $0
                2200           .EN
```

SYM to Spinwriter Hardware

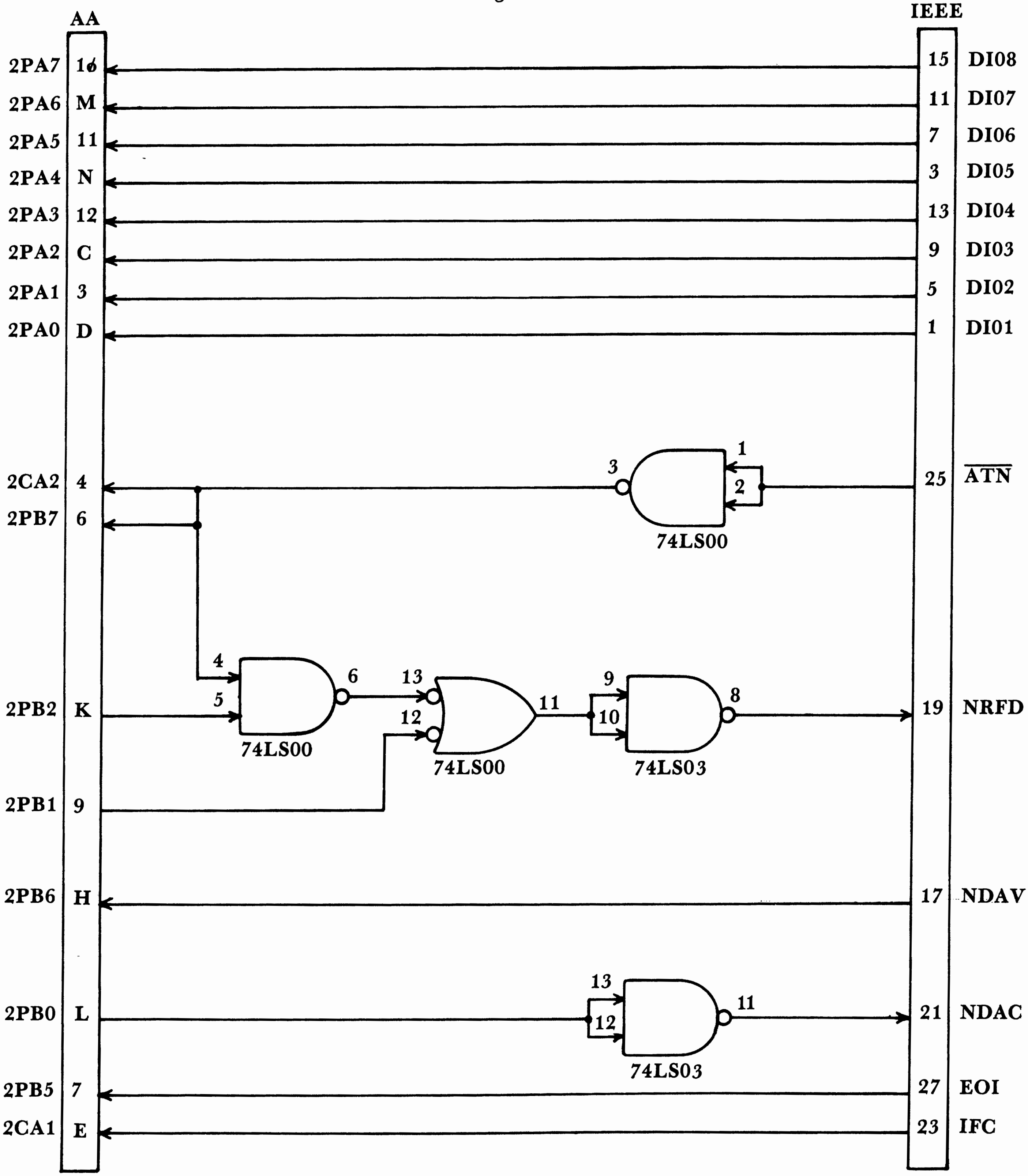


Editor's Note: For those of you who don't have issue 1, we're reprinting these two charts. RCL

TABLE 1

NAME	SET BY	DESCRIPTION
DI01-DI08	Talker	Data Input/Output. These lines carry the commands and data.
NRFD	Listener	Not Ready for Data. When low, it means the device is not ready to receive data. It is set high when the device is ready.
DAV	Talker	Data Valid. When high, it means the data on the data lines is not valid. It is set low once all NRFD goes high and valid data has been placed on the data lines.
NDAC	Listener	Not Data Accepted. When low, it means that the data has not been accepted. It is set low once DAV goes low and the data has been latched.
ATN	Talker	Attention. Signals that the byte on the DIO lines is a command.
EOI	Talker	End Or Identify. Signals that the last data byte is being transferred.
IFC		Interface Clear. Resets all devices.

Figure 1


AA
IEEE
2PA7 16
2PA6 M
2PA5 11
2PA4 N
2PA3 12
2PA2 C
2PA1 3
2PA0 D
15 DI08
11 DI07
7 DI06
3 DI05
13 DI04
9 DI03
5 DI02
1 DI01
2CA2 4
2PB7 6
25 ATN
74LS00
2PB2 K
2PB1 9
74LS00
74LS00
74LS03
19 NRFD
2PB6 H
17 NDAV
2PB0 L
74LS03
21 NDAC
2PB5 7
2CA1 E
27 EOI
23 IFC

©

SYM High Speed Tape

Gene Zumchak

The SYM has two different tape formats, the low speed or KIM format, and its own high speed format that can handle 185 bytes per second, which is not bad at all . . . if it works. The high speed format has given problems from the beginning. The new SYM monitor, version 1.1 was changed significantly in the tape routines to overcome the early problems. Also, newer SYMs use a different bias network on the tape input comparator and a fatter (.22 mfd) input coupling capacitor (C16). (Synertek advises that a few users have improved their tape reads by reducing C16, a typical value being .05 mfd.)

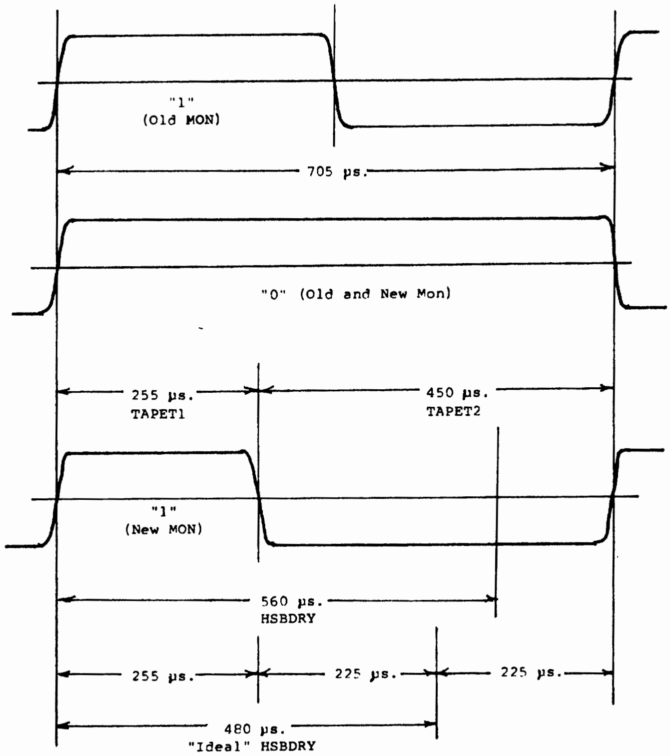
If you have an early SYM and still use the original version 1.0 monitor you won't be able to benefit from this discussion. I recommend very strongly that you obtain the new monitor. It's available from SYM Users Group, P.O. Box 315, Chico, CA 95927, for $16, and includes the resistor mod kit.

Nevertheless, even if you have the hardware mods and the new monitor, there is no guarantee that you will get reliable tape reading. The differences in success appear to be most affected by the tape recorder. Oftentime a cheap discount store recorder will give good results when a more expensive name brand unit will not. Frequency response of the recorder does not seem to be a criterion for predicting success. The SYM high speed format, and most high speed techniques depend upon measuring the time interval between transitions on the tape. Misinterpret one transition time and it's all over. The transitions are put on the tape very accurately. However, when the tape is played back, the high frequency components may experience significant phase shifting, affecting the zero crossing positions. Thus the high frequency shifting, and not so much the frequency response, appears to be the culprit. Fortunately, the new SYM monitor has some variables built into the tape routines that allow you to "tweak" the tape read/write programs to accomodate your recorder. These variables are shown in the accompanying figure, reproduced by permission of Synertek.

In the SYM format, the bit period is constant. A "one" is two transitions per bit period, and a "zero" is one transition per bit period. In the original monitor, the two intervals for the one were symmetical. In the new monitor, however, the first interval, (the only one measured) is narrower than the second, making it easier to distinguish between a short period (one) and a long period (zero). The intervals are specified by variables TAPET1 and TAPET2 which are initilized by reset to $33 and $5A respectively. These numbers represent a number of 5-microsecond intervals. Thus each bit time is $8D (141 dec.) intervals or 705 microseconds. The transition time interval is measured by starting the 6532 timer at $FF, counting down with the divide by eight clock. When a transition is detected, the value originally in location $A632 = HSBDRY (High Speed BounDRY) is added to the value from the timer. If the interval was short, the counter will not have counted down very far from $FF and adding HSBDRY will result in a carry which is interpreted as a "one-bit" transition. Thus the ability to distinguish between a one and a zero depends upon how carefully we choose the high speed boundary value. The default value of $46 (70 decimal) gives a boundary time of 70 x 8, or 560 microseconds. Synertek arrived at this value experimentally by trying several popular recorders. There is no guarantee that this value is ideal for your recorder. To split the difference between the short and long transitions would give an "ideal" boundary of 255 + 225, or 480 microseconds, or 60 ($3C) 8-microsecond intervals. If your recorder is closer to the ideal response, the default value of 560 microseconds will cause slightly narrow zero intervals to be interpreted as ones giving a bad reading. Before I took a look at the numbers, I experimentally determined the value of HSBDRY for my Panasonic recorder to be about $3C. Actually there was quite a range from $40 down to $39, but HSBDRY definitely needed to be smaller. Interestingly, I still can load tapes only over a very narrow range of volume settings.

If indeed it is the phase shifting of high frequency components that affects zero crossings, then perhaps low-pass filtering the tape output before it goes onto the tape would improve performance. Then again, I do need the tone control as high as it will go to give best results. It would seem that with the diode clipping at the input of the comparator, the tape read would be relatively insensitive to amplitude, with a high volume being ideal. However, with my SYM that is not the case. Clearly, a great deal of experimenting can be done pre-filtering tape dump output before it is recorded, and conditioning the playback output before it is decoded.

So far we have discussed only changing the value of HSBDRY to improve our read capability. However, the tape dump parameters TAPET1 and TAPET2 can also be modified. To generate SYM compatible tape, their values should not be changed radically, and their sum should equal $8D. On the other hand, if the sum is changed, the bit time and the corresponding number of bytes per second will change. We can make the tape speed faster or slower, and still read it back with the regular SYM programs by changing HSBDRY correspondingly. Just for kicks, I made TAPET1 $22 and TAPET2 $46, and was able to get fairly reliable loads with HSBDRY $30. This is a byte rate of approximately 250 bytes per second. It may be possible to double the SYM's high speed rate



NAME	LOCATION	DEFAULT VALUE
HSBDRY	$A632	$46 (70 dec.) 350 µs.
TAPET1	$A635	$33 (51 dec.) 255 µs.
TAPET@	$A63C	$5A (90 dec.) 450 µs.

and still get good loads. The important thing, however, is to get reliable loads at the regular high speed.

Unfortunatley, there are still a number of problem sources that have nothing to do with SYM hardware and software. You may be using a bad tape. Your recorder may be excessively noisy, or generate motor noise. You might suspect the latter if the Sync display indication occasionally flickers even when set at the optimum volume setting. Sometimes a capacitor (.05 to .1mfd) from the input of the comparitor (pin 3) to ground will solve this problem. To help find other problem sources, a list of guidelines, provided by Synertek, are reproduced at the article's end.

In summary, SYMMERs still having problems with tape loading and using the new monitor may only need to adjust the value of HSBDRY ($A632), thanks to Synertek's forsight in making the tape parameters variables. Remember, however, that this value, and all system RAM is initialized by RESET and will have to be fixed after each Reset.

There is certainly a lot of experimentation that can be done on the SYM high speed tape reading and writing. I hope that the information in this brief article will inspire other SYMMERs to do some investigation. I'm sure that others besides myself will want to hear about any discoveries you make.

Twenty Important Cassette Recording Guidelines

1. Use high quality tape (Maxell UD or equivalent).
2. Use shortest tapes possible. You can shorten tapes to several minutes in length if you enjoy splicing.
3. Use shielded cable between your computer and the cassette recorder.
4. Keep heads and pinch rollers clean.
5. Keep heads aligned for tape interchangability.
6. Avoid recording too close to beginning of tape.
7. Make sure cassette is properly seated in recorder.
8. If you have trouble with a cassette try another. You can have a bad spot on tape or a warped cassette.
9. Highest setting of tone control is usually best.
10. A dirty recorder volume control can cause tape dropouts.
11. Make sure cassette connection plugs make good contact.
12. Rewind cassettes before removing them from recorder.
13. Store cassettes in dust-proof containers.
14. Avoid exposing cassettes to heat or magnetic fields.
15. Before recording, wind cassette to one end and fully rewind.
16. Cassette recorders will give you problems once in a while (They don't like certain cassettes, etc.). If one gives you problems most of the time replace it.
17. Make sure that MIKE plug is connected before recording. On most recorder the TAPE light will glow while recording.
18. You may have to record with the EAR plug out for some tape recorders.
19. Always use AC adaptor with recorder for best results.
20. When a tone control is available, adjust it to the highest possible setting (maximum treble). ©

KIM Rapid Memory Load/ Dump Routine

Bruce Nazarian
1007 Wright Street #3
Ann Arbor, MI 48105

This routine works well for mass entering of stuff like long programs from a hex dump or similar, where you can tell at a glance where any errors in your entries are. A few words of additional explanation about it:

For those users who would rather have a Carriage Return activate the address entry portion and the associated functions, substitute ASCII CR ($0D) at location $010E. This will do the trick and is the same as Markus Goenner's function from his TTY load routine from K.U.N. Thanks go to him for the use of some of his programming techniques.

The directions also indicate that the program will list until it senses a key pressed at the end of a line. This is true, but the user should only use one of the DATA keys on the keypad, not ST or RS.

Finally, the routine will only indicate the stopped address after the user commands RUBOUT thru his terminal. Then the KIM monitor will print the current pointer, which will be the address where it stopped dumping.

If you want the routine to present one line of hex at a time, and wait on a key depression before looping back again and printing another line, make this change:

0147 20 6A 1F JSR KEYIN (Instead of the getkey subroutine)
014A D0 FB BNE 0147
014C EA EA NOP's to fill previous coding

```
0100                              ORG $0100
0100  D8              ENTER       CLD              Clear decimal mode
0101  A9  00                      LDA #$00         Zero out the input buffers
0103  85  F8                      STA INL          Low. . . . .
0105  85  F9                      STA INH          And High. . . .
0107  20  2F  1E                  JSR CRLF         Use KIM Subroutine to send functions
010A  20  5A  1E      ADDR        JSR GETCH        Input one character.. (of starting addr)
010D  C9  20                      CMP #$20         Check for go ahead.. (Insert 0D for CR)
010F  F0  05                      BEQ DATA         If yes, load address from buff in pointer.
0111  20  AC  1F                  JSR PACK         If no, load character into INL,INH
0114  F0  F4                      BEQ ADDR         ...and loop back again
0116  20  CC  1F      DATA        JSR OPEN         Move INL,INH, to POINTL,POINTH..
0119  20  2F  1E      DECIDE      JSR CRLF         (Saves bytes, doesn't it?)
011C  20  5A  1E      INPUT       JSR GETCH        Now input some Hex for the code...
011F  C9  4C                      CMP #$4C         'L' (Load memory)?
0121  F0  2E                      BEQ LOAD         Yes, branch to LOAD portion (0151)
0123  C9  51                      CMP #$51         'Q' (Dump from memory)?
0125  D0  F5                      BNE INPUT        No, ignore invalid characters;Loop..
0127  A9  0F          DUMP        LDA #$0F         Set up byte counter (16 decimal)
0129  8D  7F  01                  STA COUNT        stick it in $017F
012C  20  2F  1E                  JSR CRLF         New line, please..
012F  20  1E  1E                  JSR PRTPNT       Output the current pointer address
0132  20  9E  1E                  JSR OUTSP        ...and space it...
0135  20  9E  1E      GET         JSR OUTSP        ...again...
0138  A0  00                      LDY #$00         Set up Y-Register for Indirect addressing
013A  B1  FA                      LDA (POINTL),Y   Load contents of pointed address
013C  20  3B  1E                  JSR PRTBYT       ...and print as two hex digits...
013F  20  63  1F                  JSR INCPT        Increment the double-byte pointer
```

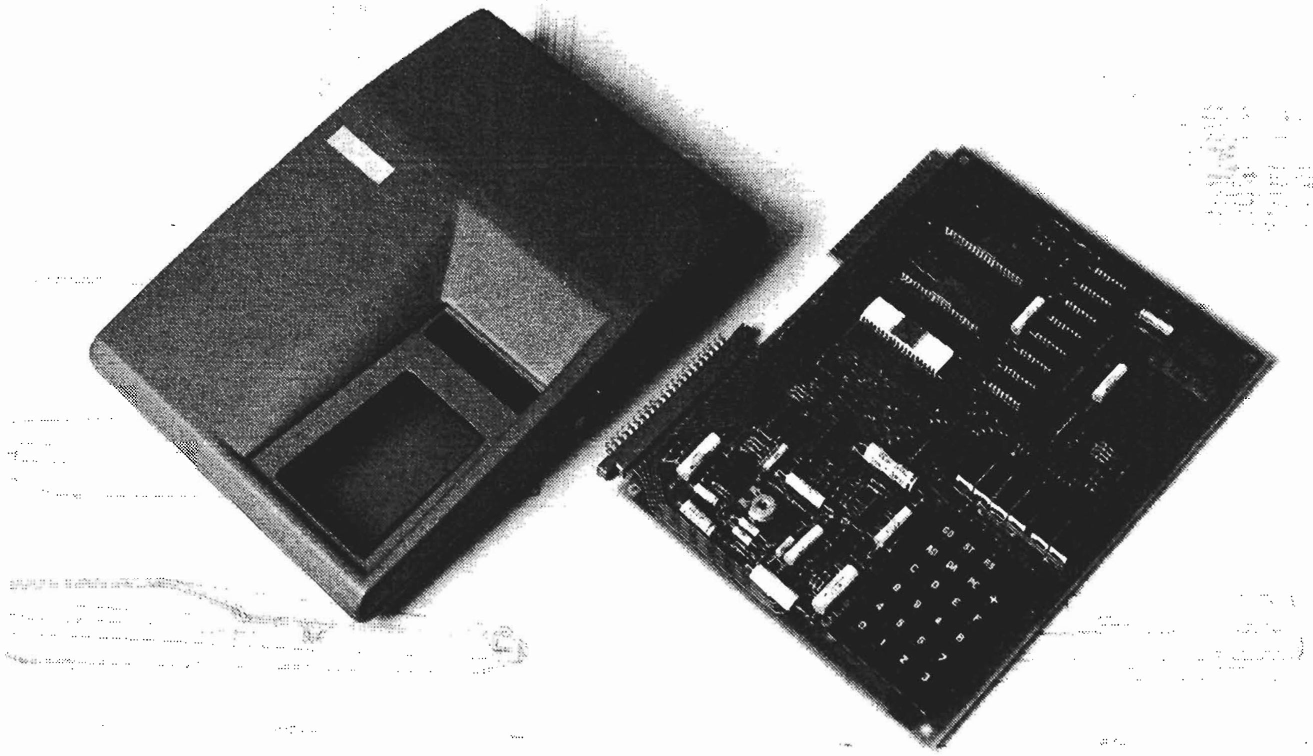
```
0142    CE   7F   01              DEC COUNT         Decrement the byte counter
0145    10   EE                   BPL GET           And loop back if not finished yet
0147    20   6A   1F              JSR GETKEY        After 16th byte, test for end of list
014A    C9   15                   CMP #$15          ...and if no key is pressed,
014C    F0   D9                   BEQ DUMP          go back and output another 16 bytes...
014E    4C   64   1C              JMP CLEAR         else jump to Clear input buffs..
0151    20   2F   1E    LOAD      JSR CRLF
0154    20   5A   1E    READ      JSR GETCH         Input one character..
0157    C9   0D                   CMP #'CR'         ..and if it is a carriage return..
0159    F0   F6                   BEQ LOAD          ..let it function, but ignore it..
015B    C9   1B                   CMP #'ESC'        ..or if it is "Escape"...go 015F
015D    D0   06                   BNE STORE         ..if not, must be valid.. Store it.
015F    20   80   01              JSR STRING        ..else send '? KIM ?' prompter...
0162    4C   64   1C              JMP CLEAR         ..and clear buffers..exit load routine
0165    20   AC   1F    STORE     JSR PACK          Pack character into INL,INH
0168    D0   EA                   BNE READ          If packed value is zero, skip it..
016A    20   5A   1E              JSR GETCH         Get second byte of Hex code
016D    20   AC   1F              JSR PACK          ..and pack it also..
0170    A0   00                   LDY #$00          Set up for indirect addressing
0172    A5   F8                   LDA INL           Bring in packed value..
0174    91   FA                   STA (POINTL),Y    .. and store it at pointed address
0176    20   63   1F              JSR INCPT         Increment the double-byte pointer
0179    18                        CLC
017A    90   D8                   BCC READ          Branch always..
017C    EA   EA   EA              NOP               Waste some space
017F    [XX]            COUNT     [This location used to hold the variable byte cntr]
0180                              ; Subroutine "STRING" to send KIM prompter
0180                              ORG $0180
0180    A2   0C         STRING    LDX #$0C          Set up X-reg as counter
0182    BD   90   01    STRNG2    LDA TABLE,X       Get character at TABLE + X
0185    20   A0   1E              JSR OUTCH         Ship it out...
0188    CA                        DEX               Decrement the counter
0189    10   F7                   BPL STRNG2        Loop is not finished
018B    60                        RTS               Else return to mainline when done
018C    EA   EA   EA              NOP               NOP's to fill

190     20   3F   20    TABLE     .BYTE 'SP,?,SP,
0193    4D   49   4B              M,I,K
0196    20   3F   00              SP,?,NUL,
0199    00   0A   0D              NUL,LF,CR
019C    0D                        CR'
```

Some Instructions To Help It All Make Sense:

1. This routine is set up for an I/O device of the user's choosing, as long as it is fed thru the KIM internal TTY port.. Users with other I/O will have to modify the coding to suit their particular situation.
2. The routine is self-contained on Page One and leaves all other memory free for user programs, but be prepared, as always, to re-read the routine from cassette should the stack overwrite the routine.
3. Execute as follows:
   After loading the coding, a "GO" executed at address $0100 will get the ball rolling.. your terminal should immediately execute a CR/LF sequence and will pause... Begin by typing in the four digit address you wish to start loading, or dumping from.. If you err in typing, just correct by typing in the correct address again, just like the KIM TTY monitor.. A "SPACE" after the correct address is in place will enter that address into the pointer.. The program will again send CR/LF and pause.. now, enter "L" if you wish to use the rapid load routine, or "Q" if you wish a formatted memory dump from your indicated address.. If LOAD was chosen, you may now begin entering data in two-digit HEX and the pointer will be taken care of for you automatically.. a good way to do this is

QUICK CHANGE ARTISTRY



ENGINEERED SPECIFICALLY FOR THE KIM-1 MICRO COMPUTER
- Protection of Chips and Other Components
- Viewing Angle of Readout Enhanced
- Improved Keyboard Position for Easier Operation

EASILY ASSEMBLED
- Absolutely No Alteration of KIM-1 Required
- All Fasteners Provided
- Goes Together in Minutes with a Small Screwdriver

ATTRACTIVE FUNCTIONAL PACKAGE
- Professional Appearance
- Four Color Combinations
- Improves Man/Machine Interface

MADE OF HIGH IMPACT STRENGTH THERMOFORMED PLASTIC
- Kydex 100*
- Durable
- Molded-In Color
- Non-Conductive

AVAILABLE FROM STOCK
- Allow Two to Three Weeks for Processing and Delivery
- No COD's Please
- Dealer Inquiries Invited

TO ORDER: 1. Fill in this Coupon (Print or Type Please)
2. Attach Check or Money Order and Mail to:

NAME ______________________

STREET ______________________

CITY ______________________

STATE ____________ ZIP ____________

Please Ship Prepaid ______ SKE 1-1(s)
@ **$24.50** Each
California Residents please pay
$26.09 (Includes Sales Tax)

enclosures group

771 bush street / san francisco, california 94108

Color Desired blue ☐ beige ☐
black ☐ white ☐

* TM Rohm & Haas

Patent Applied For

to enter two hex digits, and then space, as the routine will ignore the packed space character and only enter the valid hex... If DUMP was chosen, the routine will now commence to dump the contents of memory consecutively from your indicated address like this:

```
0200 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0210 EA EA EA ............................. etc.
```

IT WILL LIST CONTINUOUSLY UNTIL YOU PRESS A KEY ON THE KIM KEYPAD AND HOLD IT DOWN AT THE END OF A LINE.. It will then stop and indicate the stopped address.

©

KIM-1 Tidbits

Harvey B. Herman
Chemistry Department
University of North Carolina at Greensboro
Greensboro, N.C. 27412

I have been using KIM for a number of years and wish to share programs which I have developed or modified with the readers of Compute II.

The first item is a modification to the KIM tape verify program from Issue #13 of 6502 User's Notes. This program has a small bug which affects TTY use. The TTY delay characters (CNTL30/CNTH30) are stored in $17F2 and $17F3 and are overwritten by a section (VEB) of the original verify program. Instead of the comforting KIM message on completion of the program, all I got was a meaningless chugging. The following program (origin $300) circumvents the problem by shortening the VEB section so the delay characters remain intact. I now include this in KIM Microsoft BASIC, as the User program, so I can check tapes after a SAVE.

Item 2 is a modification to KIM Microsoft BASIC (serial number 9011) which allows one to append programs on tape to the current one (if any) in memory. Line numbers must be higher in the appended program and cannot overlap. Otherwise the only noticeable change is that one must remember to NEW before LOAD when appending is not desired. I have found this very helpful in conjunction with a renumbering program, written in BASIC (see 6502 User's Notes no. 13, p. 12).

I hope these programs will be found useful and plan to share other tidbits with Compute II readers in the future.

```
                   0100 ;
                   0110 ;KIM TAPE VERIFY PROGRAM
                   0120 ;
                   0130 ;HARVEY B. HERMAN
                   0140 ;
                   0150                .BA $300
                   0160                .OS
                   0170 CHKL           .DE $17E7
                   0180 CHKH           .DE $17E8
                   0190 VEB            .DE $17EC
                   0200 LOAD12         .DE $190F
                   0210 LOADT9         .DE $1929
0300- D8           0220 VERIFY         CLD
0301- A9 00        0230                LDA #$00
0303- 8D E7 17     0240                STA CHKL
0306- 8D E8 17     0250                STA CHKH
0309- A2 06        0260                LDX #$06
030B- BD 16 03     0270 LOADP          LDA PROG-1,X
030E- 9D EB 17     0280                STA VEB-1,X
0311- CA           0290                DEX
0312- D0 F7        0300                BNE LOADP
0314- 4C 8C 18     0310                JMP $188C
0317- CD 00 00     0320 PROG           .BY $CD $00 $00
031A- 4C 1D 03     0330                .BY $4C $1D $03
031D- D0 03        0340 PATCH          BNE FAILED
031F- 4C 0F 19     0350                JMP LOAD12
0322- 4C 29 19     0360 FAILED         JMP LOADT9
                   0370                .EN
```

```
                   0100 ;
                   0110 ;APPEND MODIFICATIONS TO
                   0120 ;KIM MICROSOFT BASIC
                   0130 ;SERIAL NUMBER 9011
                   0140 ;
                   0150 ;HARVEY B. HERMAN
                   0160 ;
                   0170                .BA $2785
                   0180 ;ADJUST TAPE LOAD POINTERS
2785- 38           0190 NEWLOAD        SEC
2786- A5 7A        0200                LDA *$7A
2788- E9 03        0210                SBC #$03
278A- 8D F5 17     0220                STA $17F5
278D- A5 7B        0230                LDA *$7B
                   0240 ;NAIVE HARVEY
278F- B0 02        0250                BCS SKIP
2791- E9 00        0260                SBC #$00
2793- 8D F6 17     0270 SKIP           STA $17F6
                   0280 ;ORIGINAL CODE CONTINUES
                   0290                .BA $2744
                   0300 ;ASSIGN ID 01 TO TAPES
2744- A9 01        0310                LDA #$01
                   0320                .BA $2026
                   0330 ;POINTER TO NEWLOAD
2026- 84 27        0340                .SI NEWLOAD-1
                   0350                .EN
```

©